National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file   Votre référence

Our file   Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

UNIVERSITY OF ALBERTA

A Computational Approach to Conceptual Design of Mechanical Systems

by

Masoud Shariat-Panahi ©

A Thesis submitted to the Faculty of Graduate Studies and Research in partial fulfilment of the requirements for the degree of **Doctor of Philosophy**.

Department of Mechanical Engineering

Edmonton, Alberta
Fall 1995

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE DU DROIT D'AUTEUR QUI PROTEGE SA THESE. NI LA THESE NI DES EXTRAITS SUBSTANTIELS DE CELLE-CI NE DOIVENT ETRE IMPRIMES OU AUTREMENT REPRODUITS SANS SON AUTORISATION.

ISBN 0-612-06285-6

Canadä

UNIVERSITY OF ALBERTA

RELEASE FORM

Name of Author:	**Masoud Shariat-Panahi**
Title of Thesis:	**A Computational Approach to Conceptual Design of Mechanical Systems**
Degree:	**Doctor of Philosophy**
Year this degree granted:	**1995**

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

M. Shariat Panahi

Masoud Shariat-Panahi
4-9 Mechanical Engineering
University of Alberta
Edmonton, Alberta
Canada
T6G 2G8

Date: Sept. 30, 1995

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **A COMPUTATIONAL APPROACH TO CONCEPTUAL DESIGN OF MECHANICAL SYSTEMS** submitted by **MASOUD SHARIAT-PANAHI** in partial fulfilment of the requirements for the degree of **DOCTOR OF PHILOSOPHY**.

Reginald Toogood

Dr. R. W. Toogood (Supervisor)

O R Fauvel

Dr. O. R. Fauvel (External Examiner)

Dr. D. R. Budney

M Lipsett

Dr. A. W. Lipsett

Rao

Dr. M. Rao

Date: 6 October 95

To my parents,
who taught me how to learn

ABSTRACT

The process of *Mechanical Design* roughly consists of three distinct, yet interactive phases: *Functional Design, Conceptual Design* and *Parametric Design.* In *Functional Design,* a perceived need is systematically broken down and formally presented as a set of standard mechanical functions. In *Conceptual Design,* an artifact (mechanical system) which can perform those functions is visualized in the form of an arrangement of "generic" physical elements. Finally, in *Parametric Design,* the (optimal) values of the design parameters of these generic elements are determined, and a detailed description of the artifact is generated.

The fact that the engineering community's perception of the nature of the first two phases is still in its infancy has impeded serious attempts at automating *Mechanical Design* as an integrated process. The problem is commonly blamed on the lack of realistic models of *Functional-* and *Conceptual Design* that both explain the mechanisms of their work and can be implemented in computer code.

This research is devoted to the development and computer implementation of a comprehensive model of Mechanical Conceptual Design. The model builds on the assertion that optimal designs may be obtained through a step-wise transformation of an initial functional description of a device to a structural description provided that

a) at each step, values of the behavior-defining parameters of the device are calculated so that the functional behavior of each component and its contribution to the satisfaction of the initial requirements can be fully determined, and

b) throughout the design process, all *feasible* design alternatives are nurtured and preserved for a final choice-stage. This is to avoid the possible loss of the optimum design due to a best-first search strategy during the design process.

The computer implementation of the model has led to the development of a conceptual design system which finds a number (≥ 1) of feasible arrangements of existing mechanical elements for a given set of initial functional requirements. The special architecture of the system allows for the contribution of the various objectives of the artifact's lifecycle to the design process, according to the notion of *concurrent design.* A computational approach to constraint analysis, using Genetic Algorithms, has been implemented which reduces the need for user intervention and eliminates biases stemming from user-dependency.

ACKNOWLEDGEMENTS

First and foremost I would like to thank Dr. R.W. Toogood, my supervisor, for his patience, advice and support throughout the course of this research. His kindness and scholarly attitude provided a warm and stimulating environment for me to work in. I also wish to thank Dr. D.R. Budney and Dr. A.W. Lipsett for serving as my committee members and Dr. O.R. Fauvel, of University of Calgary, for his valuable comments on this thesis.

I am grateful to the Department of Mechanical Engineering, University of Alberta, NSERC and the Iranian Ministry of Culture and Higher Education for their financial support of this work. Especially I am indebted to Dr. J.D. Dale, whose thoughtfulness made me feel at home at the U of A.

Last, but not least, I would like to thank my family for their support and encouragement during this long haul.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION AND OVERVIEW 1

1.1 THE NATURE OF DESIGN 1

1.2 *MODELS* OF THE DESIGN PROCESS 1

1.2.1 DESCRIPTIVE MODELS 2

1.2.2 PRESCRIPTIVE MODELS 2

1.2.3 COMPUTER-BASED MODELS 4

1.3 WHAT IS A *COMPUTER-BASED* MODEL? 6

1.4 THE THREE STAGES OF MECHANICAL DESIGN 7

1.4.1 FUNCTIONAL DESIGN 8

1.4.2 CONCEPTUAL DESIGN 8

1.4.3 PARAMETRIC DESIGN 9

1.5 A TAXONOMY OF COMPUTER-BASED MODELS OF DESIGN 11

1.6 COMPLICATIONS OF MODELLING THE *MECHANICAL CONCEPTUAL DESIGN* 12

1.7 CLASSIFICATION OF MECHANICAL DESIGN PROBLEMS 14

1.8 SCOPE OF THE DISSERTATION 15

1.9 REPRESENTATION ISSUES 17

1.9.1 STANDARD ELEMENTAL FUNCTIONS 21

1.9.2 EXAMPLE 22

1.10 SUMMARY 23

CHAPTER 2: RELATED WORK 25

CHAPTER 3: A COMPUTER-BASED MODEL OF MECHANICAL CONCEPTUAL DESIGN 45

3.1 THE BASIC IDEA 45

3.2 DESIGN BY EXPLORATION (DbE) 47

3.2.1 THE INITIAL REQUIREMENTS 49

3.2.2 FUNCTION-TO-PHYSICAL ELEMENT MAPPING 50
3.2.3 EXPLORING THE ELEMENTS' POTENTIALS 51
3.2.4 UPDATING THE FUNCTIONAL REQUIREMENTS 54
3.2.5 SETTING UP/UPDATING THE SEARCH TREE 57
3.2.6 VERIFYING PARTIAL DESIGNS 59
3.2.7 ITERATING THE PROCESS 60
3.3 CONVERGENCE AND COMPLETENESS CONSIDERATIONS 67
3.4 SUMMARY 68

CHAPTER 4: IMPLEMENTATION OF "DESIGN BY EXPLORATION" 70
4.1 PROBLEM REPRESENTATION 71
4.2 SEARCH/REASONING STRATEGY 72
4.3 THE *BLACKBOARD* MODEL OF PROBLEM SOLVING 77
4.4 *CONCEPTUAL DESIGNER:* THE SYSTEM ARCHITECTURE 83
4.4.1 THE BLACKBOARD 84
4.4.2 THE KNOWLEDGE SOURCES (KS's) 89
4.4.3 THE CONTROL MODULE 92
4.4.4 THE USER INTERFACE 94
4.5 *CONCEPTUAL DESIGNER:* THE ACTION CYCLE 98
4.6 SUMMARY 101

CHAPTER 5: PROBLEM-SOLVING TECHNIQUES 103
5.1 THE *NOMINATION* KNOWLEDGE-SOURCE 103
5.2 THE *EXPLORATION* KNOWLEDGE-SOURCE 109
5.2.1 PROBLEM FORMULATION 111
5.2.2 PROBLEM EVALUATION 118
5.2.3 GENERAL SOLUTION STRATEGY 128
5.2.4 RE-FORMULATING THE PROBLEM 130
5.2.5 THE SOLUTION SCHEME 132
5.2.5.1 GENETIC ALGORITHMS 133

5.2.5.2 *TUNING* THE GENETIC PARAMETERS FOR DbE . 143
5.2.5.3 CONVERGENCE ISSUES 147
5.2.5.4 PARTITIONING THE EQUALITY SUBSET 154
5.2.5.5 THE SOLUTION SCHEME AT A GLANCE 158
5.3 THE *EVALUATION* KNOWLEDGE-SOURCE 166
5.4 THE *VERIFICATION* KNOWLEDGE-SOURCE 173
5.5 SUMMARY AND CONCLUDING REMARKS 176

CHAPTER 6: APPLICATION EXAMPLE 181
6.1 *DESIGN BY EXPLORATION* REVISITED 181
6.2 AN ELEVATOR FOR THE DISABLED 183
6.2.1 INITIAL SPECIFICATIONS, ASSUMPTIONS AND CONSTRAINTS 184
6.3 INITIAL REPRESENTATION OF THE PROBLEM 184
6.4 FORMAL REPRESENTATION OF THE PROBLEM 187
6.5 GENERATING CANDIDATE DESIGNS 188
6.5.1 NOMINATING CANDIDATE COMPONENTS 188
6.5.2 EXPLORING CANDIDATE COMPONENTS 189
6.5.3 UPDATING THE FUNCTIONAL SEARCH TREE 192
6.5.4 VERIFYING THE CANDIDATE ALTERNATIVES 194
6.6 CRITICIZING THE PARTIAL DESIGNS 195
6.7 UPDATING THE (STRUCTURAL) SEARCH TREE 198
6.8 ITERATING THE DESIGN PROCESS 199
6.9 COMPARISON OF RESULTS WITH THOSE OF HUMAN DESIGNERS 220

CHAPTER 7: SUMMARY AND CONCLUSIONS 260

BIBLIOGRAPHY 264

GLOSSARY 285

APPENDIX A: A DIRECTORY OF THE *STANDARD ELEMENTAL FUNCTIONS* 289
APPENDIX B: STRUCTURAL REPRESENTATION OF MECHANICAL SYSTEMS . . 296
APPENDIX C: TYPICAL SEARCH PATTERN OF THE GENETIC ALGORITHM 301
APPENDIX D: INDEX TO THE COMPONENTS-DATABASE 306
APPENDIX E: TEST PROBLEMS TO DETERMINE OPTIMUM VALUES OF THE GENETIC PARAMETERS 311

LIST OF FIGURES

1-1 The *Design-Evaluate-Redesign* meta-model 9

1-2 A single *function block* 18

1-3 Interconnections in a functional diagram 18

1-4 A typical FBD hierarchy 21

1-5 FBD of a simple drill 23

2-1 Typical "function symbols" and their equivalents 26

2-2 The building blocks of a "schematic description" 27

2-3 Some of the component-icons used in "schematic synthesis" 28

2-4 The current-meter (a) and one immediate solution (b) 29

2-5 Two schematic solutions for the current-meter problem 29

2-6 The "specifications graph" of a spur-gear drive 31

2-7 The transformed specification graph of a spur-gear drive 31

2-8 Functioning of the *synthesis strategy* 32

2-9 The system architecture of the "concept modeller" 34

2-10 A welded cantilever beam example and its solution steps 36

2-11 General IDIDE problem-solving strategy 38

2-12 General structure of a *Function Logic Diagram* 39

2-13 The function logic diagram of a mousetrap 40

2-14 A typical function-form mapping of an automobile, based on the VROOM model . . . 41

2-15 Classification of mechanical design problems according to Zhang and Rice 42

2-16 Pre-defined configuration of an overhead-crane 43

3-1 The *Design by Exploration* model 48

3-2 The requirements FBD of a drill 49

3-3 Three matching elements for the 1st elemental function 50

3-4 Instances of the three candidate elements 54

3-5 Updated requirements for elec. motor and I. C. engine 56

3-6 Updated requirements for elec. gear-motor 56

3-6 Updated requirements for elec. gear-motor 57

3-8 Search tree of the drill; first level 58
3-9 Updated search tree of the drill; first level 60
3-10 FBD of a hypothetical element 60
3-11 Updated requirements diagrams of partial designs 61
3-12 Partial search tree of the drill 63
3-13 Pruned search tree of the drill 64
3-14 Pruned search tree of the drill 66
4-1 A simple problem representation 71
4-2 Forward search; first step 72
4-3 Forward search; partially complete tree 73
4-4 Backward search; first step 74
4-5 Backward search; partially complete tree 75
4-6 The *blackboard* model of problem-solving 78
4-7 A simple blackboard system for the example 79
4-8 Hypothetical solution state for the example 80
4-9 The solution state after first activation of the design exper. 81
4-10 The solution state after completion of the fourth level 81
4-11 The optimal design generated by the optimization expert 82
4-12 System architecture of the Conceptual Designer 83
4-13 Typical contents of the Structural-Representation Panel 85
4-14 Typical contents of the Functional-Representation Panel 87
4-15 A simple *procedure* 89
4-16 Partial rule-base of a "size" expert module. 90
4-17 Typical organization of a *procedure* knowledge-source 91
4-18 Typical organization of a *rule-base* knowledge-source 92
4-19 Part of the *design manager*'s rule-base 93
4-20 User interface: components and communications .94
4-21 Sample FBD of initial requirements 95
4-22 How the user interface reformats equality constraints 97
4-23 Sample graphical presentation of a desig 98

5-1 The *nomination* knowledge-source . . . 104
5-2 Contents of a typical *component cell* . . . 106-108
5-3 The operation of the *nomination KS*. . . . 109
5-4 The *exploration* knowledge-source . . . 110
5-5 Operation of the *exploration KS*. . . . 111
5-6 The *assignment* algorithm . . . 123
5-7 Internal constraints of the spur-gear drive . . . 124
5-8 Initial requirements of the gear design problem . . . 125
5-9 Initial incidence matrix of the spur-gear design equations . . . 125
5-10 Incidence matrix; assignments and free variables . . . 127
5-11 The reduced (square) incidence matrix of the spur gear drive . . . 128
5-12 The inequality constraint subset (**G**) . . . 129
5-13 Augmented equation set of the spur-gear drive . . . 131
5-14 Typical generation zero for the box-design example . . . 134
5-15 Raw and relative fitnesses of the population of Figure 5-14 . . . 135
5-16 Weighted roulette wheel for generation zero . . . 136
5-17 *Mating pool* for generation zero . . . 136
5-18 Evolution of populations in successive generations . . . 137
5-19 *Crossing over* of strings 1 and 4 in Figure 5-17 . . . 137
5-20 Mutation of a gene . . . 138
5-21 Results of a reproduction iteration . . . 139
5-22 Generation 1 (reproduced from generation zero, Figure 5-15) . . . 139
5-23 Basic operation of *Genetic Algorithms* . . . 141
5-24 Results of tests to find optimal value of CR . . . 145
5-25 Generation-dependent MP . . . 146
5-26 Rebooting mechanism for a dead-locked search . . . 150
5-27 Probability of optimal convergence vs. number of trials . . . 151
5-28 Modified GA with rebooting mechanism . . . 152
5-29 Efficiency of GAs compared to other search techniques . . . 153
5-30 Partitioning a sparse matrix . . . 156

5-31 Incidence matrix (SIM) of an equation set 157

5-32 First two steps in transforming SIM to NEWSIM 158

5-33 Incidence matrices after four transactions 159

5-34 The block-triangularized incidence matrix NEWSIM 160

5-35 Operation of *Exploration Module* 161

5-36 The ordered (block triangular) incidence matrix of the spur gear drive 162

5-37 Gear drive's inequality constraint subset (**G**) 162

5-38 Discretized ranges of free variables 164

5-39 Feasible solution alternatives 165

5-40 The *evaluation* knowledge-source 166

5-41 The operation of *evaluation* knowledge-source 167

5-42 Drill design problem, presented by its requirements FBD 170

5-43 Drill design problem: an intermediate solution state 170

5-44 FBD of the gear-drive instance 171

5-45 Drill design problem: one iteration later 172

5-46 A hypothetical FBD 174

5-47 Two candidate solutions 175

5-48 Operation of *verification* knowledge-source 177

6-1 Functional description of the elevator drive system 186

6-2 Formal representation of the initial requirements 187-188

6-3 Nominated components to perform the first elemental function 190

6-4 Feasible instances of the candidate components - first level 191

6-5 First level of the functional search tree - updated nodes 192-193

6-6 The *environment* knowledge source 195-196

6-7 Feasible partial design alternatives at the first level of the search tree 200

6-8 Knowledge-pool of the cable-hoist (partially shown) 201

6-9 The constraint set of the spur-gear drive 203

6-10 Multiple solutions to the spur-gear sub-problem 204

6-11 Alternative ways to further reduce the speed of the electromotor's shaft 205

6-12 Partial search tree 206

6-13 The two solutions to the speed-adjustment sub-problem 209
6-14 Results of the expansion of the worm-gear node 209
6-15 Solutions to the speed-adjuster/cable-hoist sub-problem 211
6-16 Solutions to the speed-adjuster/chain-hoist sub-problem 211
6-17 Solutions to the hydraulic-drive sub-problem 212
6-18 Solutions to the pneumatic-drive sub-problem 213
6-19 Results of the expansion of the rack-and-pinion node 213
6-20 Results of the expansion of the power-screw node 214
6-21 Pruned search tree after expanding the first leaf-node 215-216
6-22 Typical partial designs with half-stroke hydraulic/pneumatic drives 217
6-23 Final design alternatives 225-259
6-24 Number of major components in each of the final design alternatives 221
6-25 Student-group design ideas for the elevator - first group 223
6-26 Student-group design ideas for the elevator - second group 224
B-1 Graphical and schematic representation of four typical components 297
B-2 Graphical representation of two subsystems 298
D-1 Index to the components-database 307-310

CHAPTER 1
INTRODUCTION AND OVERVIEW

1.1 THE NATURE OF DESIGN

The research reported here is concerned with *Design Methodology,* rather than *design* itself. While the latter deals with the question of "*what* to design" to satisfy certain requirements, the former is primarily concerned with the question of "*how* to design", that is, how to model/teach/aid/automate the process of finding answers to the question "what to design". In other words, the objective of research in design methodology is not a product, but rather the knowledge of how to design products.

Whereas design itself dates back perhaps to the early days of man on earth, design methodology as the scientific study of the design processes is relatively new. In a mature field, the research community will share a common view of the appropriate research methodologies, the unsolved problems in the field and the criteria for appraising a proposed methodology. In the emerging field of design research, especially mechanical design, no such consensus exists. This lack of consensus both causes some chaos and makes the design field exciting, as it promises that new, revolutionary paradigms will emerge.

A scientific study of the design process may seem to be the logical way out of this dilemma. However, it has been argued (Cross, Naughton and Walker 1987; Simon 1969) that "scientific study of the design process" is a contradiction in terms, for "science" is the systematic study of natural entities whereas "theories of design" study artificial entities, i.e. those constructed by man. Nonetheless, design activity has been investigated both as a natural process and as an artificial process. The motive behind the former has been to improve descriptive- as well as prescriptive models of human design processes, and the goal of the latter has been to develop mathematical and computational tools for aiding/automating design (Kannapan and Marshek 1992).

Whether we consider research in design methodology a "science" or "exploring a technological activity" (Jacques and Powell 1980), it is believed that it is still in a pre-theory stage. Research approaches needed for generating design theories are much less well-understood than the classical scientific methods for evaluating theories.

1.2 *MODELS* OF THE DESIGN PROCESS

Attempts to shed some light on the enigmatic nature of design and to postulate paradigms for it can be roughly categorized in three groups: those trying to understand and explain the way design is currently being carried out (by humans), those prescribing how it should ideally be carried out, and those building on potentials of the computer technology. These three groups of attempts have led to the development of what we respectively call *descriptive*, *prescriptive* and *computer-based* models of design.

1.2.1 DESCRIPTIVE MODELS

"Descriptive" models of design are based on the belief that by "watching" eminent designers "design" and then scrutinizing and systematizing their thinking route, one can gain insight into the complex process of design and can formulate a paradigm which will enable others to design equally successfully. With this in mind, many researchers from various fields have studied the question of "how humans design", that is, what processes, strategies and problem-solving methods they use to generate a design. They have conducted protocol studies on both novice and expert designers solving trivial as well as complex problems (Adelson and Soloway 1984; Ullman and Dietterich 1986; Ullman, Stauffer and Dietterich 1987; Waldron et al. 1987; Wallace and Hale 1987; Schon 1988; Marples 1961) to determine how they approach the problems in each case and thus to learn what causes excellence in design.

Cognitive models of design are the best examples of descriptive models. They have been generated and nurtured by a school of thought that believes "design systems must be based on human design processes" (Gero and Coyne 1985). These models are intended to foster computer programs that "simulate or emulate the skills that humans use as they solve problems" (Laird, Newell and Rosenbloom 1989). Aiming at this, a number of attempts have been made to exploit the results of protocol studies along with artificial intelligence techniques to develop intelligent CAD systems that undertake some aspects of design the way humans do (Adelson 1989b; Laird, Newell and Rosenbloom 1989; Maher 1987; Newsome and Spillers 1989).

1.2.2 PRESCRIPTIVE MODELS

Another group of researchers postulate that design should follow a certain logical pattern, regardless of whether or not this pattern is currently exercised by experienced designers. The various "patterns" proposed by this group are normally referred to as "prescriptive" models of design. Prescriptive models can be divided into two categories: those that prescribe how the design process ought to proceed and those that prescribe the characteristics that the design artifact ought to have. One major class of prescriptive models of the design process was initiated in 1950's in Germany by Hansen and his colleagues (Bischoff and Hansen 1953; Bock 1955; Hansen 1956). This family of models is now known as *systematic design* models. A large body of research has since been published on systematic design (Hansen 1965; Hansen 1974; Koller 1976; Rodenacker 1984; Roth 1982) in German, of which only two books have been translated to English (Hubka 1980; Pahl and Beitz 1991). The essence of these models, according to Rodenacker (Rodenacker and Claussen 1974) is the following steps (from Pahl and Beitz 1991).

- Clarify the task (requirements of the problem),
- Establish the function structure (the logical relationships satisfying the requirements) using functions derived from three main functions: *separate*, *connect* and *channel*,
- Choose the physical process (the physical relationships to satisfy the requirements),
- Determine the embodiment (the constructional relationships to satisfy the requirements),
- Check the logical, physical and constructional relationships by appropriate calculations,
- Eliminate disturbing factors and errors,
- Finalize the overall design,
- Review the chosen design,

We will explain some of these steps later on in this report.

A standard description of what the design process should be recurs in much of the engineering-education literature, especially in design textbooks. A review of the more popular mechanical-design textbooks, including those by Shigley (1986), Deutschman (1975), Pahl and Beitz (1991), Edwards and McKee (1991), Asimow (1962) and Jones (1972) shows that, despite minor differences, the following steps characterize a common trend in prescribed models of design.

- Recognition of need
- Specification of requirements
- Generation of design ideas (concept level)
- Choice of the best idea
- Detail (embodiment) design
- Evaluation of the design

The pioneers of prescriptive models of the designed artifact (as opposed to the design process) are Suh and Taguchi. Suh's Axiomatic Design (Suh 1990) asserts that the best design is one that **(a)** maintains the independence of its functional requirements (FRs) and **(b)** has the minimum information content (i.e. the information necessary to meet the functional requirements). Consider, for example, the problem of designing a refrigerator door (from Suh's book). There are two functional requirements for such a device: to access the contents of the fridge and to provide an insulated enclosure to minimize the energy loss. In an ordinary fridge-door (horizontally opened), these two FRs are coupled: each time the door is opened, the cold air escapes the fridge. The design is therefore inferior as it does not allow the two FRs be satisfied independently. On the contrary, a vertically opened door (like the one used in chest freezers) would allow access to the contents of the fridge with virtually no energy loss, since the cold air tends to stay down and not to escape through the opening on the top. The latter, therefore, is considered superior to the former design.

The second axiom implies that the less information needed for manufacturing, maintaining and using a product, the better its design is. According to this axiom, a design that (fulfils the first axiom and) has fewer components, uses more standard components rather than special ones, performs only the required functions, and has less geometric irregularities (such as asymmetries and exceptional tolerances) is a superior design.

Taguchi (Taguchi 1978; Taguchi 1987; Taguchi and Wu 1980), on the other hand, proposes only one criterion for measuring the goodness of a design: *robustness*. A "robust" design is one that minimizes the *quality loss* over the life of the product, where "quality loss" is defined as the deviation from the desired performance of the product. Whereas the axioms of Suh assume that a good design is one that meets a set of well-designed functional, structural and economical goals, Taguchi is concerned with the sensitivity of a design to unpredictable factors that may arise in manufacturing and use. He defines "noise factors" as the undesirable, uncontrollable and costly factors that cause a product to deviate from its functional target values. He then uses statistical techniques, especially design of experiments, for parametric design and tolerance specification.

1.2.3 COMPUTER-BASED MODELS

The third major group of design models are *computer-based* models. Unlike the other two groups, namely the descriptive and the prescriptive models, computer-based models do not try to offer an *ideal* plan for design, nor do they denounce or avoid the plans suggested by other design models. Here the objective is solely to present a method by which a computer may accomplish a design task. The development of computer-based models of design has been fuelled by the developments in computer science and technology. The fact that computers outperform humans in certain areas such as massive calculations and large-scale data processing has inspired researchers to exploit these capabilities to develop computer systems for designing or assisting in design. These computer systems, which were initially generated on an application-by-application basis, later led to the development of a number of design models.

A computer-based design model may in part be derived from observation of how humans perform the task, but this connection is not necessary, as the former may in turn suggest prescriptions for human processes (Finger and Dixon 1989). As a matter of fact, computer-based models should not be considered "alternatives" to descriptive and prescriptive models, but rather "computer adaptations[1]" of them with modifications to improve their performance and to adjust them to the existing computer technology. To illuminate this statement and to explain how we believe computer-based models can cultivate their prescriptive counterparts, we mention two observations made by researchers who performed comprehensive studies of human design behaviors.

Juster (1985) cites several authors (Hykin and Lansing 1975; Tebay, Atherton and Wearne 1984) who studied the performance of many designers and found that the design process they follow is different from that suggested by prescriptive models. He attributes this variance both to the designers who are not systematic enough and to the proposed models that are unrealistic in their assumptions about the order lines of the process. This problem will never arise in the case of computer-based models, firstly because a computer is always "systematic enough" to follow a model given to it and to not "defy" it, and secondly because computer-based models are usually "extracted" from programs that have already proven working, and this leaves no room for them being "unrealistic".

The other observation made by various researchers concerns creativity and originality in design. From his study of a number of design projects, Marples (1961) concludes that "designers reuse familiar solutions and will not explore alternatives or innovative ideas unless their design fails badly and cannot be salvaged". Also Ullman et al. (Ullman and Dietterich 1986; Ullman, Stauffer and Dietterich 1987) observe that designers pursue a single design concept and tend to patch and repair that single concept until they arrive at a feasible design, if at all, rather than generating new alternatives. This avoid-new-ideas-as-much-as-possible strategy does not conform to the implications of prescriptive models at all, as the spirit of those models is to generate multiple ideas and evaluate them in order to spot the best one. Again this problem can be largely avoided by incorporating multiple-design-generation strategies into the computer-based models.

To close this discussion, we quote Dixon (1988) who concludes that cognitive studies of the design process alone are unable to provide theoretical foundations for design, as "they involve far too many ill-defined variables to support a theory", and that "prescriptive models are premature until they can be based on a validated theory". He further believes that "computer-based studies, if used appropriately to discover and explain the knowledge and strategies needed for design, could lead to the desired theoretical foundations for the design process".

---

[1] We use the term "adaptation" or "free adaptation" and not "implementation" to distinguish between those computer systems that try to transform the exact contents of a model into a computer program and those that devise independent computer systems inspired by the ideas embedded in that model.

In the rest of this work we shall be reporting on the development and implementation of a special computer-based model of the mechanical design process. Therefore, we will need to state more clearly what we mean by a "computer-based model" and whether belonging to the area of "mechanical design" has any implications on its development.

1.3 WHAT IS A *COMPUTER-BASED* MODEL?

In a 1985 article titled "Who Said Robots Should Work Like People" (Seering 1985), Seering attacks the then-popular idea of constructing robots that look and work like humans. He argues that the idea of making "an army of robots with humanoid features that would work in American factories day and night, undercutting the advantages of cheap labor abroad ... was fatally flawed because it assumed that humans are optimally designed to perform manufacturing tasks and therefore deserve to be emulated". He then points out some of the human weaknesses in manufacturing compared to the machines and asserts that "robots modeled after humans share all these inherent weaknesses". He concludes that in order to take full advantage of the robot technology, we need to make robots that work like machines not like people, and on things *they* are good at, rather than things *people* are good at.

The same argument is true of *computers* and *design methodologies*. It is indisputable that humans are "the best" when it comes to *creative design*, but they are not nearly as efficient as computers when it comes to the tedious, repetitive calculations involved in routine design. Many computing sciences- as well as engineering design researchers have tried to generate tools that would enable computers to emulate the human process of design. But these attempts will soon prove to be misdirected, and hence fruitless, for computers would function most efficiently when they design as computers, not as humans.

For one thing, optimal design(s)[2] can generally be achieved only through generation and evaluation of multiple feasible alternatives, rather than by sticking to a single idea and patching and repairing it; because no matter how effective these "repairs" are, they will not mutate a design into a different one. As discussed earlier, humans often overlook this actuality and consequently lose some good design alternatives, and so will computers if they are programmed to follow humans' footsteps. Computers, on the other hand, have the ability to repeat a specified design procedure over and over at very little or no extra cost, and thus to generate multiple design alternatives (provided, of course, that the computer code contains an element of random-selection to avoid repetetive results). It will be then easy to introduce an element of evaluation/optimization to the procedure or let the user pick the optimal design based on some specified criteria.

---

[2] The term "optimal design" here refers to "the best (perhaps out of several acceptable) plans for satisfying a set of requirements" and not to a design whose values of parameters have been optimally "tuned" using numerical optimization techniques.

Now that we have somewhat specified what a computer-based methodology is not (should not be), let us explain what it is. *A computer-based model of a design process is a formalization of the process that can be embodied in a computer program capable of performing the corresponding design tasks.* We distinguish between computer processes that design (i.e. make design decisions) and those that assist in designing by analyzing a design or providing graphical and computational assistance. In this work we are concerned with the former, that is, the computer systems that perform a complete or partial design process and the models these systems are based on.

As mentioned earlier, being "computer-based" does not imply that a model is thoroughly isolated from descriptive and prescriptive models. The model may partially coincide with either or both of those, but this relation is not necessary. For example, a computer-based model may prescribe that after the multiple solution alternatives are produced, Suh's axioms (Section 1.2) be employed to determine the goodness of each design. The main distinction between the two genres of models is hence that a computer-based model builds its strategies on the known capacities of the computers and therefore *can* be implemented in the form of a computer system. Let us emphasize here that our definition of a so-called "intelligent" or "automated" system is limited to what we discussed in this section. We strongly believe that truly intelligent systems that can emulate the creativity and intellectual activities of humans are beyond the reach of the existing technology.

In order to discuss the computer-based models in the area of mechanical design (the subject of this work), we first need to outline the mechanical design process and the stages involved in its implementation. This is because computer-based models of mechanical design commonly pertain to specific stages of the process and a "universal model" to represent the entire design process does not yet exist.

1.4 THE THREE STAGES OF MECHANICAL DESIGN

Several researchers have studied the classification of the design tasks and have proposed *taxonomies* of mechanical design problems in particular (Aylmer and Johnson 1987; Chandrasekaran 1988; Dixon et al. 1988; Matsuta and Uno 1980; Sambura and Gero 1982; Yoshikawa 1982). A survey of these classifications and the ones suggested by the descriptive and prescriptive models of design shows that the core of the process is believed to comprise three distinct, yet interactive stages: *functional design, conceptual design* and *parametric design.* Each stage is distinguished by the states of knowledge about the entity to be designed, before and after the stage is performed. This, of course, does not encompass such pre- and post-design stages as "specification of the need" and "manufacturing", as they fall into other categories of production activities. The three stages just mentioned represent, in sufficient detail, the essence of the mechanical design process, although some researchers have proposed more (or less) detailed classifications. These stages are further discussed in the following sub-sections.

1.4.1 FUNCTIONAL DESIGN

Mechanical design, in the most general sense, begins with the realization of a "need" or a set of requirements. Functional Design is then the transformation from these abstract requirements to the functional description of a mechanical system that will satisfy those requirements. This formal description will be generated through systematically breaking up the abstract requirements and expressing them hierarchically in terms of detailed functional building blocks that are closer to elementary physical functions. The resulting functional description must be precise, yet solution neutral, and must follow a standard format so that it can be communicated between various designers/computer systems. Without loss of generality, we shall assume that the functional descriptions of mechanical systems are presented in the form of *Function Block Diagrams*, described in Section 1.9.

1.4.2 CONCEPTUAL DESIGN

Conceptual Design is often defined as the transformation from a functional description of an artifact to a structural description of it, where the former is (in our case) in the form of a function block diagram (Section 1.9) and the latter in the form of a configuration of *component-types*. The term "component-type" refers to generic mechanical elements (such as "bevel gear", "thrust roller bearing" and "helical tensile spring") whose values of design/performance parameters are yet to be specified.

With this definition of conceptual design, it might seem trivial to carry out: just take the functional description of a device and replace each functional component of it with its physical equivalent. In practice, however, this will not work (at least not feasibly). While structure and functioning of a machine are intimately related, they do not *uniquely* determine each other.

Generally, in mechanical design, a single function can be performed by various structural configurations and a single component can contribute to more than one elemental function. In addition to their primary, intended functions, mechanical elements exhibit some secondary, incidental behavior which might or might not be desired in a particular design. This unintended behavior has then to be compensated for/exploited in order to result in an acceptable degree of design integration and compactness. A direct, one-to-one transformation of the initial functional requirements to physical components normally will not result in good mechanical designs. This issue will be further discussed later in this chapter.

The final product of Conceptual Design is a (number of) configuration(s) of component-types which will satisfy all the specified requirements. This information is then passed to the next phase, the Parametric Design.

1.4.3 PARAMETRIC DESIGN

Parametric Design is the transformation from a generic structural description of an artifact to specific instance(s) of its structure. In this phase, attributes of the component-types selected through Conceptual Design are quantitatively specified and values of the corresponding design parameters are determined. These values can be numeric or non-numeric (e.g. a material type or a catalog number). Optimization techniques may be employed here to find those parameter values that best meet a specified criterion. The output from Parametric Design, as the last stage of the design process, is the detailed (structural) description of one or more arrangement(s) of component-instances which collectively satisfy all the initial requirements.

Ideally, this concludes the mechanical design process. In practice, however, the design thus generated has yet to be checked for consistency and overall feasibility. This is because the additional information generated in the Parametric Design phase, especially when component optimization is involved, might reveal some discrepancies in the overall design or impose new constraints which may or may not be readily satisfied by

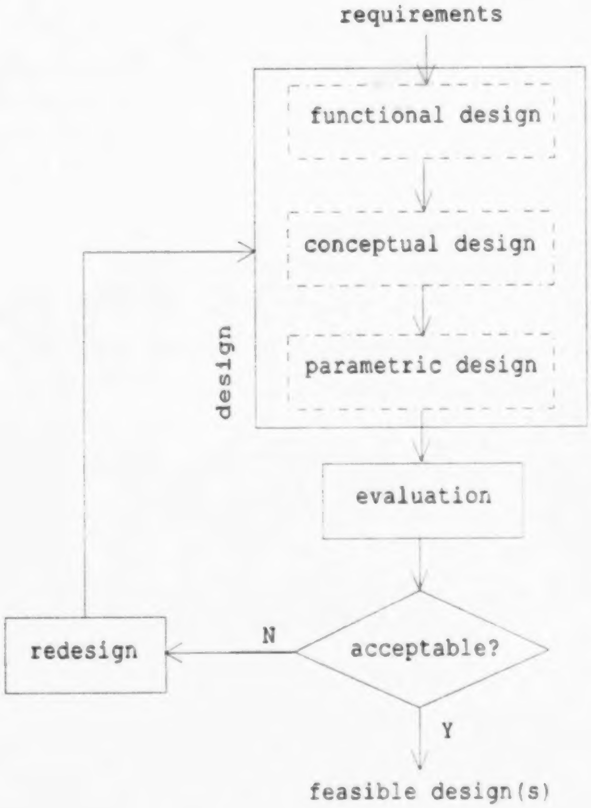


Figure 1-1: The *Design-Evaluate-Redesign* meta-model

other components. If a design checks successfully, it will be accepted as a feasible solution to the design problem. Otherwise, the machine must be completely or partially redesigned to meet the original/ augmented requirements.

The strategy outlined in the last paragraph above is a rough description of the *Design-Evaluate-Redesign* meta-model (Figure 1-1) shadowing over virtually all design models presented earlier. There is nearly a consensus that once a design is initially generated according to any paradigm, it needs to be evaluated and, if necessary, redesigned until it meets all the requirements of the problem. Once this is achieved, the description of the finalized design(s) can be passed on to the next stage, namely the manufacturing stage.

Occasionally, the manufacturing agents find a design hard or infeasible to manufacture. Then the design, along with the manufacturing engineer's comments, is returned to the designer for further modifications. The cycle goes on until both parties are satisfied with the design. Furthermore, the same problem may arise when the product reaches the maintenance stage. Then yet another set of modifications needs to be done to the design of the product. This dilemma is typical of the standard one-task-at-a-time production strategy which prescribes each stage of the process (e.g. design, manufacturing and maintenance) to be considered separately with no interactions between them.

In order to avoid this costly procedure, an alternative approach, known as *Design for X*, has been proposed where *X* represents any of the life-cycle objectives of a product such as *manufacturing, maintenance* and *reliability*. In practice, however, most of the work done on this strategy has focused on *Design for Manufacturing*. In essence, the strategy requires that the manufacturing information be incorporated into the design process and "manufacturing requirements" be treated as part of "design requirements". In this way a *feasible design*, if found, would be one that satisfies not only the functional (design) requirements but also the manufacturing requirements, and therefore would not receive any "surprises" in the manufacturing stage.

The ultimate form of the design for X strategy is the notion of *Concurrent Design* whereby concerns of *all* lifecycle objectives are considered during the design of a product. A more detailed discussion of concurrent design will be presented in Chapter 3.

It has been suggested (Fauvel 1992; Fauvel, Gu and Norrie 1993; Yin 1993) that in order to build a sound framework for concurrent design, one should identify and take into account the three information domains that are the "currency of design". These three domains are the *environment* domain, the *function* domain and the *embodiment* domain. We shall refer to the implications of this suggestion in the following chapters.

1.5 A TAXONOMY OF COMPUTER-BASED MODELS OF DESIGN

After this brief introduction to the various stages of design, we can now categorize the computer-based models of the mechanical design process developed/proposed so far. Nearly every design model reported in the literature is specific to one of the design stages just introduced, and the few paradigms postulated for the entire design process are rather sketchy and have yet to show potential for implementation. There are models of functional design, conceptual design and parametric design. Some researchers (for example see Finger and Dixon 1989) have counted a fourth group of models called *configuration design* models. These are the models that propose a plan for configuring a set of pre-determined components to satisfy certain requirements. We, however, do not make such a distinction and rather consider "configuration design" part of the conceptual design stage, as we shall discuss later on.

Without a doubt, *parametric design* is the most mature design stage of all in terms of the number of models developed for it. Functional design is the least explored area and conceptual design lies somewhere in between. In fact, nearly all of what we teach in engineering design courses is "analysis" or "parametric design". Normally the part where we talk about how to transform a need to a working system amounts to a small percentage of the contents of a design course. For the rest we teach how to quantitatively evaluate the performance of a device of given specifications (analysis) or how to determine the values of design parameters of a component to result in certain values of its performance parameters (parametric design).

Among the models of mechanical parametric design are Dixon's *Dominic* (Dixon et al. 1987; Howe et al. 1986) and *Dominic II* (Orelup, Dixon and Simmons 1988), Brown and Chandrasekaran's *DSPL* (Brown and Chandrasekaran 1986) , Shah, Ramachandran and Steinberg's *DPMED* (Ramachandran, Shah and Langrana 1988; Shah, Ramachandran and Langrana 1987; Steinberg 1987), Nicklaus' *Engenious* (Nicklaus, Tong and Russo 1987), Kannapan and Marshek's *Design Diagrams* (Kannapan and Marshek 1991) and Agogino's *Symbolic Computations* (Agogino and Almgren 1987a;b). Although these models differ significantly in complexity and versatility, they all share a quest for the values of a component/subsystem's design parameters for a given set of specifications.

Models of *mechanical conceptual design* are the subject of the next chapter and will be elaborated therein. As for *mechanical functional design* there have been, to our knowledge, very few attempts at developing a computerized system that can translate a highly abstract function (need) to an arrangement of standard, specific sub-functions. This is not unexpected, for here we are dealing with the most "intelligent" part of an activity (design) which itself is considered a highly intelligent activity of mind. As a matter of fact, functional design marks the border between routine design and invention as it tries to answer the question "what steps must be taken to satisfy a certain need". If the need has been satisfied in the past, it is conceivable to "implant" this experience in computer's memory and to make it adjust the solution for a given set of requirements. However, in case of an unprecedented need it would take a certain level of creativity to come up with a solution, which computers currently lack.

With this distinction in mind, we now proclaim that except for the overly abstract work of Dyer, Flowers and Hodges (1984; 1985), no other attempts at modelling the *invention* activity has been accomplished. As for the routine functional design, works of Hundal (Hundal 1988; Hundal and Byrne 1989; 1990), Kota and Lee (1990) and members of the German school of *Systematic Design* such as Roth (1982), Rodenacker (1984) and Pahl and Beitz (1991), though still at a preliminary stage, are about the only sources available to date.

1.6 COMPLICATIONS OF MODELLING THE *MECHANICAL CONCEPTUAL DESIGN*

In the rest of this work, we shall focus on the development and implementation of a model of *mechanical conceptual design.* While conceptual design has been relatively successfully modelled and automated in such areas as electrical and software engineering, it has not been so fortunate in mechanical engineering. This is mainly because mechanical designs have a number of special characteristics normally not found in other engineering areas. They are often composed of tightly coupled, multifunctional components[3] where not only the interactions between various components, but also the relations between the form and the functions of individual components contribute to the overall behavior of the design. A change in the *form* (shape, dimensions, material, etc.) of a mechanical component will most likely affect its *function(s)*.

It is therefore essential that both the interactions and the form-function relations of various components of a device be examined *during* the conceptual design phase, and not be completely left for a later "parametric design" phase. Design systems that fail to take this necessity into account will likely result in poor or even infeasible designs. No such complexity applies to other, more explored areas of engineering design.

Another important aspect of a mechanical system is that its overall function is determined not only by its embodiment (configuration of physical elements), but also by the interaction between this embodiment and the environment surrounding it.

---

[3] Throughout this report, we shall be using the terms *"component"* and *"element"* interchangably.

Moreover, according to the notion of concurrent design, the final product of mechanical design must simultaneously meet a variety of requirements including cost, quality, manufacturing and maintenance requirements. This means that a multitude of constraints from various sources, rather than just the functional requirements, have to be satisfied at every stage of the design process, including conceptual design. Again, this type of obligation is specific to mechanical design in general and mechanical conceptual design in particular. In coping with this problem, human designers usually rely on their experience and try to solve the problem on an ad-hoc basis.

These characteristics of mechanical devices have impeded the successful modelling and computer implementation of mechanical conceptual design in its general form. Earlier in this chapter we mentioned that an alluring, yet simplistic approach to performing conceptual design of mechanical systems is to apply a one-to-one mapping between the spaces of the required functions and physical elements. (For brevity, we shall refer to this approach as the *direct substitution* approach henceforth.) While this strategy may (and does) work for some areas of engineering design, it generally fails in mechanical design.

Design systems based on the direct substitution approach essentially fail to comply with the requirements pointed out in the last two paragraphs. Also they are flawed because they assume that each component performs only one function and therefore a different component must be provided for each functional requirement. This is definitely not true of many of the commonly used mechanical components that perform multiple functions. Systems based on this assumption will normally result in inferior or even infeasible designs.

As an example, suppose that we want to design a simple drill. Also suppose that the drill has already been functionally designed and we have learned that the overall function of the drill can be broken up into the following sub-functions.

{*convert energy (electrical to mechanical rotational), adjust rotational speed, adjust angle of axis of rotation, grip the drill bit, transfer torque to the bit*}

In a *direct substitution* approach, each of the above functions would be replaced with one physical component. A typical substitution sequence of this type could be as follows (shown here as a set of ordered pairs with the first members representing "*functions*" and the second members representing the corresponding "physical components").

{(*convert energy* → electrical motor), (*adjust rotational speed* → spur-gear set), (*adjust angle of axis of rotation* → bevel-gear set), (*grip the drill bit* → chuck), (*transfer torque to the bit* → coupling)}

Although this arrangement of components *does* perform the desired function, the component redundancies in its structure make it unjustifiable. For instance, a set of bevel gears not only could change the rotational speed of a shaft but also, because of its geometry, could provide for the desired right angle between the two gear shafts. One therefore could have replaced the two functional requirements *"adjust speed"* and *"adjust angle"* with a single component (bevel-gear set). Also, if the chuck was mounted directly on the output shaft of the bevel-gear set, it would be able to both *grip* the drill bit and *transfer the torque* to it, and we would not need a separate coupling to accomplish that. Moreover, the direct substitution approach would not care about the fact that the use of a spur-gear set will cause an unwanted offset between the input and output shafts (due to the centre distance of the gear set). This is because the approach would not take into account the implications of the gear-set's geometry on its functional behavior.

With these observations in mind, we restate our definition of mechanical conceptual design (Section 1.4.2) as *the process of visualizing an **integrated** configuration of mechanical component-types which will satisfy a set of initial requirements as determined in the Functional Design phase.* We include the term "integrated" to emphasize the significance of the two main characteristics of mechanical components, namely *multifunctionality* and *form-function interdependency*, and to highlight their implications on the design process. According to this definition, to qualify, any conceptual design methodology must provide proper tools and techniques to systematically take these characteristics into account. We shall further discuss the implications of this stipulation in Chapter 3.

1.7 CLASSIFICATION OF MECHANICAL DESIGN PROBLEMS

While different mechanical design problems share the same solution steps outlined in Section 1.4 above, the degree of creativity involved in their solution can differ greatly from one problem to another. It takes the very same steps to "invent" a device to meet a new demand as to solve a routine design problem. In both cases the requirements should be first transformed into a functional structure, then a combination of elements should be visualized accordingly and finally these elements should be parametrically designed. The difference, however, is that in the former case the functional structure has to be improvised and possibly new components need to be devised whereas in the latter, the functional structure has been already developed and successfully mapped onto an arrangement of existing components for some set of specifications.

Brown and Chandrasekaran (1986) propose a classification of design problems based on the availability of knowledge sources and problem-solving strategies for each problem. They assert that any design activity falls into one of the following categories:

- Class 1 design which leads to major inventions and for which neither the knowledge sources nor the problem-solving strategies are known in advance.

- Class 2 design which does not result in totally unprecedented artifacts, yet is not as routine as respecifying an existing device either. It normally arises in the course of routine design when a new requirement is introduced to the problem which calls for some decision making. For this class of problems, the knowledge-sources can be identified in advance, but the problem-solving steps are not exactly fixed apriori.
- Class 3 design which is basically "routine design". Here the purpose is to choose from a set of well-understood design alternatives. Typical problems of this class are "respecification" problems. For instance, redesigning a car for a different speed range involves the choice of a different engine , transmission, brakes, etc., whereas the overall design of the car remains unchanged and no new components are added.

Until recently, all computer-based models of design were devoted to the third class of problems, i.e. routine design. With the advent of new computer technology, efforts to cross the border to class 2 and rarely to class 1 have commenced. From our definition of the mechanical design stages (Section 1.4) it must be clear by now that according to the classification above, conceptual design problems fall into the class 2 problems, as there are no fixed plans to transform the functional description of a device into its structural description. However, since Brown's classification builds on fairly relaxed definitions and there is no way to accurately determine which class a design problem belongs to, we will not discuss it any further.

1.8 SCOPE OF THE DISSERTATION

We state our objective in this work as *to develop and implement a computer-based model of the mechanical conceptual design.* Unlike some other areas of engineering design, the automation of the *mechanical design (machine design)* process is still in its infancy. As mentioned earlier, only one stage of the process, namely the parametric stage, has been well explored and somewhat automated. The other stages of the process, and hence the overall process, are still far from being standardized and automated.

This shortage is usually attributed to the lack of a robust model of the mechanical design process which can be implemented in the form of working computer codes, or a *computer-based* model as per the definitions given in Section 1.2. Due to the problems partially mentioned in the previous sections, the development of a computer-based model extensive enough to represent the entire process of mechanical design has proven to be beyond the realm of existing computer technology. For this reason, researchers have come to believe that mechanical design automation is a ladder to be climbed one step at a time.

With parametric design automation reaching its maturity, we decided to focus on the next step, the conceptual design. Our goal has been to develop practical strategies and techniques to make the automation of this stage possible, and thus to pave the way for tackling the problem of automating the process in its entirety. The following definitions, assumptions and declarations will further clarify and specify the objective just stated.

- We define mechanical conceptual design as *the process of visualizing an* ***integrated*** *configuration of mechanical component-types*[4] *which will collectively satisfy a set of initial requirements.* These requirements include a description of the function(s) the device is to perform, plus any other qualitative/quantitative information/constraints applicable to the device. While the functional requirements are assumed to be given at the outset of the process, additional information and constraints may be provided at virtually anytime during the process.
- As is evident from the definition above, neither the model of mechanical conceptual design nor the computer implementation of it (as presented in this work) are intended as stand-alone entities. In the chain of mechanical design activities, *conceptual design* is the link between *functional design* and *parametric design.* The input to conceptual design is the *functional description* of the device produced by the "functional design" stage and the output from it is the generic *structural description* of the device to be passed on to the "parametric design stage".
- Without the loss of generality, the *functional description* of a device is assumed to be in the form of a Function Block Diagram composed of standard basic functions, as discussed in Section 1.9.
- The domain of application of the presented model (called *Design by Exploration* or DbE for short) comprises those devices that can be described as *configurations of existing, known mechanical elements.*
- The model and the computer system based on it are *case-independent*, that is, they are not restricted to designing only certain devices (as is the case with, for instance, expert systems). The DbE model can conceptually design any mechanical system that meets the conditions outlined in the rest of this section.
- The model does not require that specific information be initially available. It considers virtually any set of initial specifications and tries to resolve cases of over- and under-specification and find the solution(s) to the problem.
- In the present work we advocate the notion of *design automation.* Strategies involved in the implementation of the proposed model are designed to minimize or, wherever possible, eliminate the need for user intervention. The user, however, will

---

[4] In the rest of this chapter we shall refer to this configuration as the "device"; meaning the mechanical system to be designed.

be able to intervene and make design decisions if required. The model's computational approach to design decision making will lead to less-biased solutions as a result of reduced user-dependency.

- The model allows for the generation of *multiple* feasible design solutions to a given set of requirements rather than a single "best solution". This strategy will help dispel prejudice on part of the designer which sometimes precludes unconventional but superior designs. Nonetheless, the model allows for the optional use of an evaluation mechanism to choose the optimal solution out of multiple alternatives.
- The model supports the concept of *concurrent design*. It accommodates the idea of multiple knowledge-sources each representing one of the product's lifecycle objectives and each being able to evaluate/criticize the design at virtually any point during the design process.
- It is assumed that the implications of the embodiment-environment interaction can be represented either as a set of constraints included in the problem's overall constraint set, or in the form of a knowledge-source capable of evaluating the design at any point.

Each of the statements above will be further discussed in the following chapters.

1.9 REPRESENTATION ISSUES

Formal representation of the function and the physical structure of a device is a fundamental component of any attempt at design systematization. Without a formal, uniform and standard representation, an artifact cannot be properly communicated between designers and manufacturers, or even designers themselves. In this section we introduce a method for representing devices functionally. The reason for our inclusion of the topic in this introductory chapter is that the functional representation of devices and the corresponding terminology will recur in the rest of this work and a minimum familiarity with the definitions presented here is essential to understanding the contents of the following chapters. The method to represent the physical structure of devices is discussed in Appendix B.

As mentioned earlier, in this work we assume that the initial requirements of a design problem are mainly presented in the form of a Function Block Diagram (FBD). In other words, the functional description of a device to be designed is represented by a FBD, which is the final product of the functional design stage. This assumption, however, is merely a matter of convenience and has no significance when it comes to the performance of the model. Any other representation method with the same qualifications as FBDs will do equally well. These qualifications include the use of a standard vocabulary of mechanical elemental functions and the specification of interrelations between these functions (both discussed below). Since we shall be frequently referring to FBDs and their characteristics throughout this report, here we present a description of the method.

The representation method described here is based on a modified version of (IDEF$_0$) (U.S. Air Force 1981), an information representation methodology developed by the U.S. Air Force in 1981. While IDEF$_0$ was intended to model a variety of systems "including any combination of hardware, software and people," the FBD methodology is primarily intended to represent a physical system functionally. A Function Block Diagram is a structured graphical representation of the function(s) of a system. It comprises a set of boxes (nodes) interconnected by a set of arrows (arcs). Boxes represent the functions carried out by the system and arrows represent the flow of data (objects and information) that interrelate those functions.

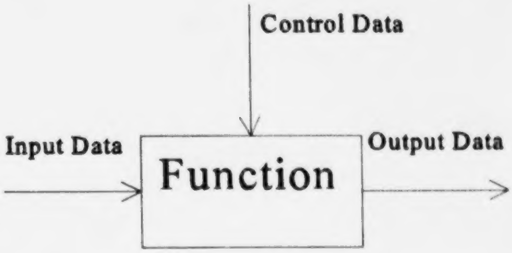


Figure 1-2: A single *function block*

The position at which an arrow enters or leaves a box conveys the specific role of the interface (Figure 1-2). The arrows entering the left side of a box represent the input data needed to perform the function and/or acted upon by the function. The arrows leaving the right side of a box represent the output data generated by the function. Finally, the arrows entering the top of a box represent the control data (conditions or circumstances) that govern the function. There can be any number of input/output/control arrows connected to a box.

Briefly then, the arrows connected to the left and right sides of a function box refer to *what* is done by the function, whereas the ones connected to the top of the box hint at *why* or *when* the

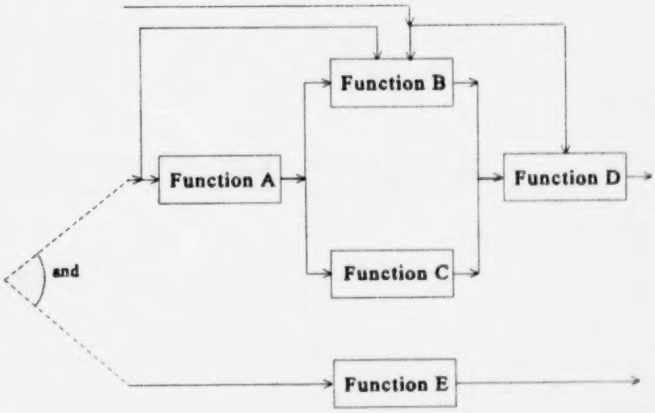


Figure 1-3: Interconnections in a functional diagram

function is triggered. Unlike the original $IDEF_0$ model, FBDs do not include *mechanism* arrows, and the bottom side of the function blocks remain unused. This is because at this stage (beginning of the design process) the structure of the device is not known. As a matter of fact, determining the mechanism(s) to carry out the required function(s) is the very goal of conceptual design.

Arrows may branch, indicating that the same data is needed by more than one function, or they may join, implying that the same class of data may be generated by more than one function (Figure 1-3). For example, an output arrow representing *mechanical energy* as the output from the function box "generate mechanical energy" (say, by an I. C. engine in a car) can divide into two branches to feed the two function boxes "displace air" (e.g. in a radiator fan) and "convert mechanical energy to electrical energy" (e.g. in a dynamo). Also, the two arrows representing mechanical energy from, say, an electric motor and an I. C. engine can join to represent the input to a double-input gearbox.

In complex designs where multiple groups of functions (rather than individual functions) are to be carried out, the overall FBD is usually formed by the junction of several sub-diagrams connected by "and" joints. One such junction is shown in Figure 1-3. Different sub-diagrams in a FBD represent different batches of functions where the performance of one batch does not affect the performance of others directly, and batches can be carried out in parallel. For example, in a tea-bag making machine one sub-process is to deliver the tea to the measuring device, measure it, and pour it in pre-made bags; whereas a parallel sub-process is to feed the special paper, measure it, cut it, make the bag and deliver it to the filling station. These two sub-processes will be shown by two separate, parallel sub-diagrams joined by an "and" joint up to the point where the bags are filled with tea. From there on, the filled bags will be the subjects of a single process.

Associated with each function block (but normally not shown in the diagram) are a number of *specifiers*. Specifiers are basically a pre-defined subset of the performance parameters of a function. The values of the specifiers are needed for the relationship between the input- and the output data of a function block to be specified quantitatively. For example, the function "Force Amplification" has "force" both as its input and output data, and has one specifier *Force Amplification Factor (FAF)*. A "FAF" value of 5 for instance would mean that the output force has a magnitude five times bigger than that of the input force.

As another example, consider the function "Rotational-Speed Change" with its input quantity "IRS" (Input Rotational Speed) and output quantity "ORS" (Output Rotational Speed) and the two specifiers "RSR" (Rotational Speed Ratio) and "TP" (Transmitted Power). Again, the set of values (RSR=3.0 and TP=10 kw) would indicate that 10 kw of mechanical power is transmitted to an output shaft rotating three times faster than the input shaft.

Besides their specifiers, Function Block Diagrams may also have associated with them a number of constraints (equality and/or inequality) representing the additional requirements that could not be contained in the diagrams or expressed in terms of the function-specifiers. These constraints mainly comprise relations that either restrain the values of the specifiers (rather than giving their values) or describe some desired structural characteristic of the designed device. An instance of the former is the relation ($2.0 \leq RSR \leq 4.0$) in the example of the last paragraph, indicating that any "Rotational Speed Ratio" between 2.0 and 4.0 inclusive is acceptable in the example transmission. As for the latter (a relation describing a desired structural characteristic of the designed device), consider the relation ($\text{Weight}_{\text{Total}} \leq$ 15 kg) which restrains the total weight of the required transmission device to 15 kg. Briefly then, the quantitative information associated with a FBD is composed of two types of relations: those that directly give the values of some of the specifiers and those that otherwise restrain the values of the specifiers and/or other design/performance parameters of the problem.

Complex mechanical systems may be represented as *hierarchies* of function block diagrams rather than a single diagram. In this case, each function block in a higher level diagram will be represented by a separate, more detailed FBD further down the hierarchy. This way, each lower level FBD will reveal a certain degree of details about its parent FBD (Figure 1-4).

In an integrated design environment, the description of objects (mechanical systems/subsystems) must be communicable among various phases of the design process as well as various designers. This implies that final (lowest level) FBDs must be composed of a common vocabulary of function blocks. One such vocabulary, called the *Standard Elemental Functions* is included in Appendix A and is discussed in the following subsection. In the rest of this report, the term Function Block Diagram will specifically refer to the most detailed version of a FBD in terms of these standard elemental functions.

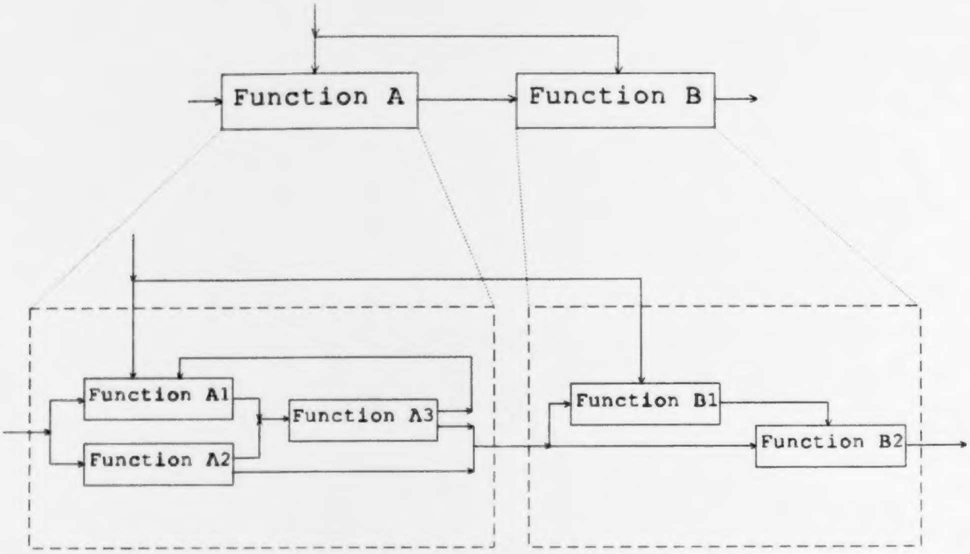


Figure 1-4: A typical FBD hierarchy

1.9.1 STANDARD ELEMENTAL FUNCTIONS

In functional design, the final presentation of the initial requirements must follow a common, formal vocabulary. An ideal vocabulary of mechanical functions has to be

- comprehensive enough to be able to describe all mechanical functions,
- case and user independent, and
- agreed upon by the engineering design community.

When decomposing a task, human designers show an experience-driven tendency to employ task-specific subfunctions. In other words, they generally stop further decomposition of a function which, they know from experience, can be performed by an existing component/subsystem. While this is definitely advantageous in traditional manual design, it is evidently in contrast with the intent of automated design which, among other goals, seeks to generate novel design ideas through impartial decomposition of desired tasks. Furthermore, the outcome of the traditional approach to task decomposition will totally depend upon the designer's personal experience and (possibly incomplete) knowledge of existing components and cannot be generally understood and processed by other designers/design phases.

Several researchers have attempted to establish a criterion for the classification of mechanical functions. A common trend has been to set up a functional hierarchy by introducing a number of "basic" functions and applying them to each of the three main quantities (material, energy and signal). Each branch in the hierarchy would be further split according to the physical form of the handled entity, input/output of the function, etc (Rodenacker 1984; Hundal 1990; Koller 1985). The number of "basic" functions varies from 3 (Rodenacker 1984) to 24 Koller (1976). In this thesis we follow a classification of our own which is based on introducing eight basic functions, refining these functions hierarchically, and applying them to various quantities. This classification is partially presented in Appendix A. Again, note that this is only a suggested set of functions intended to help us demonstrate the performance of the DbE model.

In Functional Design little attention, if at all, is paid to the solution (final design), for the objective here is just to provide a standard definition of the problem. Neither the individual block functions nor their arrangement in a function block diagram are meant to hint at particular components/subsystems, but rather to display the order in which various functions are to be carried out and their inter-relations. Later in this report we will discuss the importance of the notion of solution-neutrality in functional design and how it paves the way for possible novel design solutions.

1.9.2 EXAMPLE

Let us consider the FBD of a simple electric drill (Figure 1-5). For demonstration purposes let us assume that the output axis of the drill must be perpendicular to that of the driving motor. At the top level, the desired function is represented by a single function block (drill). It can be seen that at a control signal, the input to the block (electrical energy) is converted to a special type of mechanical energy (rotation of a drill bit at certain speed and torque). A closer look at this abstract function block reveals the details of the desired function (lowest level).

As the lower part of Figure 1-5 shows, now six *standard elemental functions* are used to describe the required drilling function. (Note that the names of the SEFs have been slightly changed from those in Appendix A so that they would graphically fit in designated boxes.) As pointed out earlier, here the arrows entering and leaving the boxes demonstrate the interrelations among the SEFs. The output signal from a "switch" function tells the "energy conversion" (e.g. by an electric motor) when to take place. The diagram also shows that the drill bit must be "gripped" (say by a chuck) before the torque is transfered to it. These two instances show the typical role of control data in the functional description of a device. We shall return to the above example in the following sections.

Function Block Diagrams are the main output from the Functional Design phase. They are then reported to, and make the input to the next phase, the Conceptual Design.

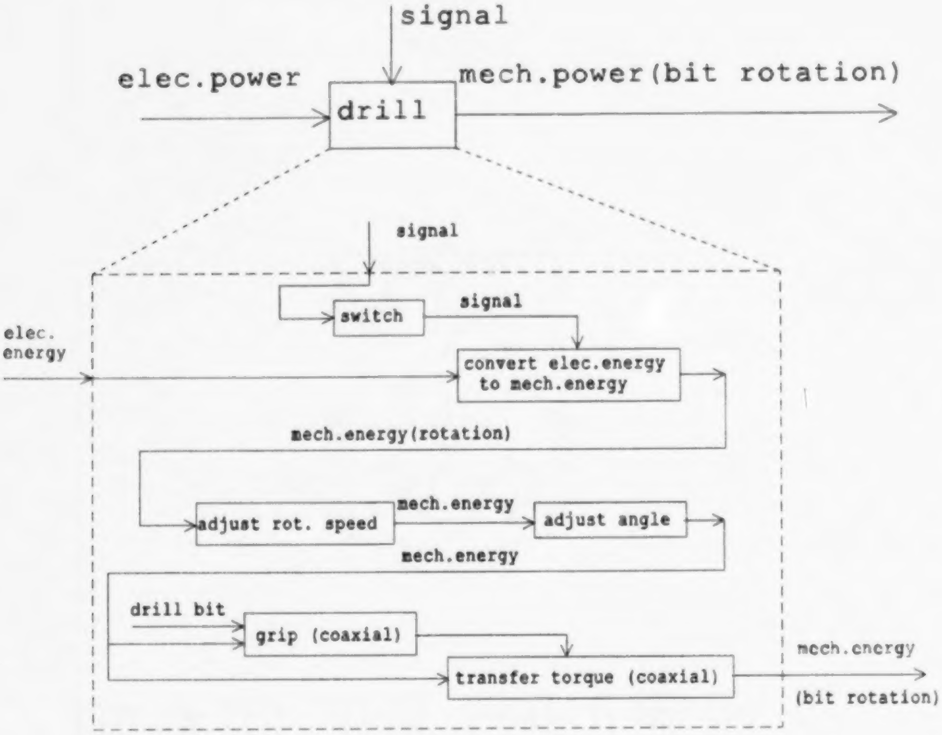

signal
elec.power
drill
mech.power(bit rotation)
signal
switch
signal
elec. energy
convert elec.energy to mech.energy
mech.energy(rotation)
adjust rot. speed
mech.energy
adjust angle
mech.energy
drill bit
grip (coaxial)
transfer torque (coaxial)
mech.energy
(bit rotation)

Figure 1-5: FBD of a simple drill

1.10 SUMMARY

In this introductory chapter we presented different views about the nature of *design* in general and *mechanical design* in particular. It was pointed out that the latter can be roughly divided into three distinct stages "functional design", "conceptual design" and "parametric design". We also cited some of the attempts at learning how humans design and whether artificial design systems could (or should) emulate that process. A review of the three major classes of proposed paradigms for design (namely the descriptive, the prescriptive and the computer-based models) was presented and it was argued that these paradigms are not "mutually exclusive".

Having pronounced the objective of the current research as *to develop and implement a computer-based model of mechanical conceptual design,* we then discussed some complications of the stated task and a number of parameters that have impeded significant developments in the field. Also, to further specify the scope of the research, we discussed the major assumptions we will be making throughout the work and highlighted some of the more important features of the developed model. Finally, we expanded one of our assumptions regarding the initial problem representation and presented an overview of Function Block Diagrams and their vocabulary, the Standard Elemental Functions.

The body of this report is organized in the following form.

- Chapter 2 presents a critical review of the related work, i.e. of the other attempts at modelling and automating the process of mechanical conceptual design to date.
- Chapter 3 provides an overview of a computer-based model of mechanical conceptual design called *Design by Exploration*.
- Chapter 4 discusses the implementation of Design by Exploration. This chapter is a bridge between the abstract overview of the model presented in Chapter 3 and the detailed description of its problem-solving tools and techniques presented in Chapter 5. It also describes the overall architecture of the Conceptual Designer, the computer implementation of the DbE model.
- Chapter 5 elaborates the problem-solving methods involved in the implementation of DbE. Computational as well as reasoning procedures are presented in detail and illustrated through example design problems, and
- Chapter 6 presents a comprehensive case study to demonstrate the performance of the *Conceptual Designer*, plus a comparison between the results produced by DbE and those produced by two groups of human designers (student projects).
- Chapter 7 contains a summary of the thesis and its contributions plus some concluding remarks.

CHAPTER 2

RELATED WORK

In this chapter we shall briefly review some of the more mature works on computer-aided conceptual design of mechanical systems reported in the literature. This will serve two main purposes: to acquaint the reader with the major developments in this area of research and to help us introduce some of the concepts that will form the basis of our discussions in the following chapters. Our account of the related research will cover only those works that, at least partially, fit into our definition of *conceptual design* in Chapter 1. This means that we shall not discuss some of the less related works that, though prominent, are concerned with other aspects of mechanical design[1], although some reviewers have not made this distinction and have referred to all these works with such general terms as "design automation" or "artificial design processes".

There will be, of necessity, limitations to our presentation of various works. Our emphasis will be on those aspects that best characterize the contribution of each work and that will highlight ours in the rest of this report. Among these aspects are *completeness, domain-independence, automation (user-independence)* and *practicality,* as will be discussed as required.

The earliest works on conceptual design and its systematization were done by Europeans, especially Germans. They started out by developing sets of icons to systematically represent mechanical functions. In doing so, they developed classifications of mechanical functions in the form of function hierarchies (Pahl and Beitz 1991; Koller 1973; Krumhauer 1974; Rodenacker 1976; Roth, Frank and Simonek 1972) and defined a symbol for each function (Koller 1976). The purpose of these works was initially to facilitate the presentation of design ideas and to help the designers "speak the same language" in design. Later, it was suggested that using a library of standard mechanisms (components), one could substitute known physical elements for functional icons to get a rough preliminary description of the device (Crossley 1980). A few examples of the "function icons" and their suggested physical equivalents are shown in Figure 2-1 (from Crossley 1980). This strategy, generally referred to as "function-to-form-mapping" in the literature, is the essence of several conceptual design systems postulated over the last decade, and will be discussed in some detail later in this report.

---

[1] This includes the valuable works of Dixon et al. (e.g. Howe et al. 1986; Orelup, Dixon and Simmons 1988), Brown (e.g. Brown and Chandrasekaran 1986) and Shah et al. (e.g. Shah, Ramachandran and Langrana 1987; Ramachandran, Shah and Langrana 1988) on Parametric Design, Erdman (e.g. Thompson, Riley and Erdman 1985; Rosen, Erdman and Riley 1987) on Linkage Design, Maher (e.g. 1985; 1987) on Configuration Design and Agogino (e.g. Cagan and Agogino 1988) on Innovative Design.

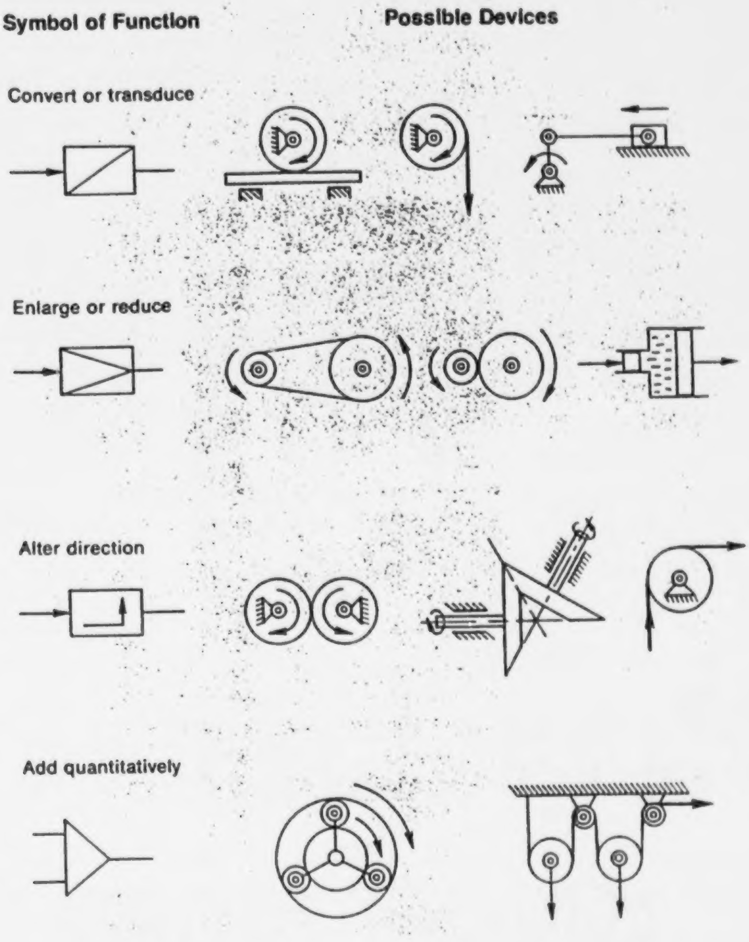


Figure 2-1: Typical "function symbols" and their equivalents
(after Crossley 1980)

Aiming at "generating designs from a specification of their abstract behavior", Ulrich and Seering (1988a; 1988b; 1989) propose a methodology called *schematic synthesis* for generating physical descriptions of a class of mechanical systems from their abstract functional descriptions. According to the classification presented in Chapter 1, their methodology spans over conceptual design as well as a special form of functional design in which the function blocks can be determined by manipulating simple governing effort-flow relations.

The domain in which they have applied their methodology is that of Single-Input, Single-Output Dynamic Systems (SISODS). This consists of the devices that can be described as networks of lumped-parameter idealized elements, and whose behavior can be specified by a relationship between a single input quantity and a single output quantity. Examples of devices in this domain include pressure gauges, speedometers, pneumatic cylinders and accelerometers. A very important characteristic of this design problem is that one need not deal with the geometry or material properties of its devices.

A conceptual design problem in this domain is then defined as follows.

"Given the input and output quantities of a mechanical system and a desired relation between the two, find the schematic description (configuration of component-icons) of a system that provides that desired relationship."

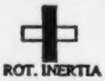
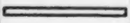
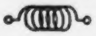
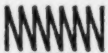
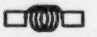
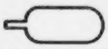
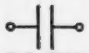
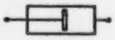
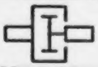
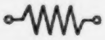
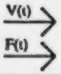
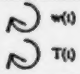
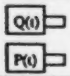
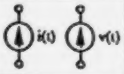
	Trans. Mechanical	Rot. Mechanical	Fluid	Electrical	Effort-Flow Relation
Inertance	 MASS	 ROT. INERTIA	 F. INERTANCE	 INDUCTOR	$E = I\frac{dF}{dt}$
Capacitance	 SPRING	 ROT. SPRING	 F. CAPACITANCE	 CAPACITOR	$F = C\frac{dE}{dt}$
Resistance	 DAMPER	 ROT. DAMPER	 F. RESISTANCE	 RESISTOR	$E = FR$
Source	 V(t) F(t)	 w(t) T(t)	 Q(t) P(t)	 i(t) v(t)	Efforts and Flows specified by these sources.
Effort Quantity	Force	Torque	Pressure	Voltage	———
Flow Quantity	Velocity	Angular Velocity	Volume Flowrate	Current	———

Figure 2-2: The building blocks of a "schematic description"
(after Ulrich and Seering 1989)

The solution strategy proposed by Ulrich and Seering is composed of two steps: 1) generate a schematic description (equivalent of a Function Block Diagram) of the system, and 2) transform this schematic description to a structural (physical) description. Since in the current work we are concerned about the latter (i.e. the conceptual design step) we shall not further discuss the former (i.e. the functional design step).

The elements used to represent the schematic descriptions in the specified domain are shown in Figure 2-2 and some of the component-icons used to represent the physical description of systems are shown in Figure 2-3. Having generated the functional description of a system, the proposed strategy initially generates a rough physical description of the system through a direct, one-to-one substitution of physical-component icons for functional icons. It then looks for opportunities to simplify this design by using what are called "physical features" of the design.

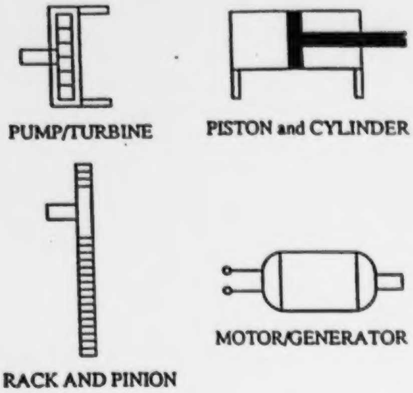


Figure 2-3: Some of the component-icons used in "schematic synthesis" (after Ulrich and Seering 1989)

The procedure just introduced can be formally presented as follows.

- Given a potential functional design, i.e. a tentative configuration of functional icons connecting the input and output quantities, find and substitute one physical icon for each functional icon.
- Using the effort-flow relations of the individual components, derive the equation relating the input and output quantities.
- Compare this equation to the desired one, that is, see if the derived equation represents the desired relationship between the input and output quantities. If so, the design is accepted and the design process ends. Otherwise
- Using the domain-specific knowledge, find out what functional elements are missing from the picture, and add their physical equivalents to the design.
- Repeat the above step until the design exhibits the desired functional behavior, or is found unmodifiable.

Figure 2-4(a) presents the example problem of a "current meter" design (Ulrich and Seering 1989) in which the input quantity is electric current and the output quantity is velocity. The desired relationship between the two quantities is "integration", that is, the integral of the output quantity (e.g. the displacement of a pointer) is to be proportional to the input quantity (current).

One immediate solution found according to the proposed strategy is shown in Figure 2-4(b). Two component-icons (an electric motor and a rack and pinion) are mounted in place of the dashed box in Figure 2-4(a). The design is considered tentative until it proves to satisfy the requirements.

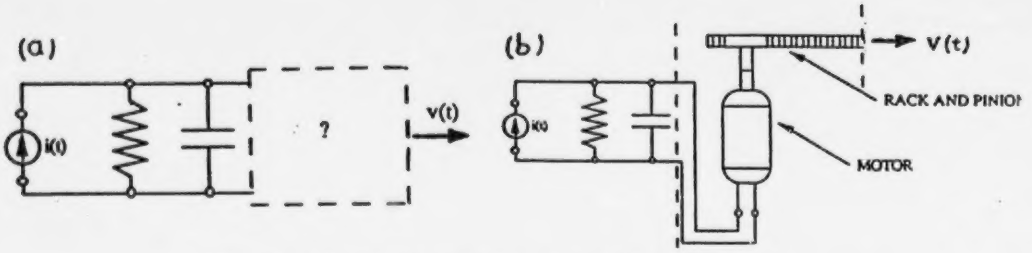


Figure 2-4: The current-meter (a) and one immediate solution (b)
(after Ulrich and Seering 1989)

The first requirement is readily satisfied: the candidate design does transform current to velocity. However, the second requirement is not satisfied, as it is the velocity itself that is proportional to the current and not its integral.

To fix this problem, a mass M and a spring are added to the system, rendering the input-output relationship one of the integral type. Two instances of the final design are shown in Figure 2-5.

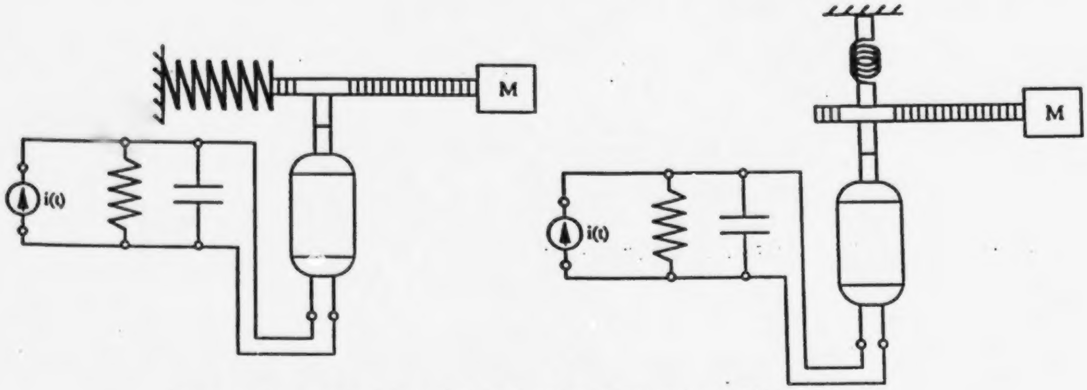


Figure 2-5: Two schematic solutions for the current-meter problem
(after Ulrich and Seering 1989)

Although successful in its task of synthesizing the physical description of a device from its functional description, the methodology has drawbacks both in its domain of application and as a general conceptual-design methodology. Among these are the following.

- The methodology is incomplete in the sense that it does not provide the means for excluding redundancies in designs and hence leaves the door open for an infinite number of design alternatives, all "acceptable" by the standards of *schematic synthesis*. In other words, the methodology approves any design that performs the desired functions, even though it may contain redundant components that do not affect the nominal behavior of the design. This way, an infinite number of designs can be generated by adding to a minimal design any of the "neutral" components or pairs of components (such as motor-generator couples).

- The methodology does not take into account the secondary functions inevitably inherent in the physical existence of the components. The components are assumed ideal, single-functioned entities with no geometric/physical significance and have no effect on the performance of the design but what they are selected for. In reality, however, the secondary functions and the physical characteristics of mechanical components often do affect the overall performance of a design and ignoring these effects will result in inferior designs, as we shall further discuss in the following chapters.
- The domain of application of the proposed methodology is limited to a very small class of mechanical systems and the techniques utilized to implement the methodology (including a mapping of the mechanical system onto the electrical domain) cannot be extended to other, broader domains.
- It is assumed that the only requirement of the problem is the desired relationship between the input and output quantities. No other constraints are taken into account and the problem is presumed static, i.e. none of the characteristics of the problem would change in the course of design. This, however, hardly represents the real-world problems, especially in the case of complex systems, where both the information and the requirements of the problem are subject to changes during the process.
- The methodology follows a general design-evaluate-refine scenario whereby no evaluation of the system is done until the design(s) is (are) complete. If partial designs were evaluated intermittently for feasibility during the design process, the infeasible ones could be detected and rejected early in the process, hence saving a noticeable amount of processing time and avoiding possible combinatorial explosions.

Perhaps the closest work to the research presented in this report is that of Hoover and Rinderle (1989). They propose a synthesis strategy for generating structural descriptions of mechanical devices initially described by their abstract functional behavior. The strength of their work is partially due to their realization of the following facts.

- Unifunctional mechanical components rarely, if at all, exist. Physical components exhibit unintended behavior in addition to their well-known, intended functions. "Form" of a component also may affect its function and must be considered simultaneously.
- Direct, one-to-one substitution of physical components for their functional counterparts will often result in inferior designs. The designer must utilize the secondary functions of the mechanical elements as well as their main functions.

The proposed strategy requires that the device to be designed be first represented by its *specification graph.* A specification graph is basically a bond-graph representation of the functional requirements of the problem. Standard function- and geometry-*primitives* are used to compose specification graphs. As an example, the specification graph of a spur-gear set is shown in Figure 2-6. In this example two primitives ("reducer" and "geometric") are used to describe the problem.

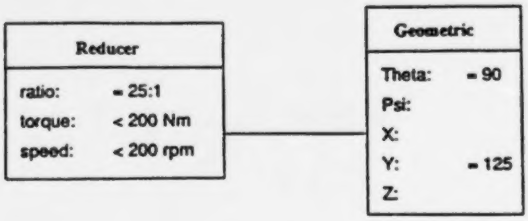

Reducer
ratio: = 25:1
torque: < 200 Nm
speed: < 200 rpm
Geometric
Theta: = 90
Psi:
X:
Y: = 125
Z:

Figure 2-6: The "specifications graph" of a spur-gear drive (after Hoover and Rinderle 1989)

Once the specification graph is generated, a series of *function-preserving* transformations is applied to the graph which eventually transforms it to a functional description of the device. A "transformation" in this context means a re-organization of the specification graph so that the resulting graph is functionally equivalent to the original one. The purpose of a transformation is to produce a version of a graph in which each building block corresponds to an actual mechanical component. To define a *function-preserving* transformation, Hoover and Rinderle differentiate between the "function" and the "behavior" of a device. In their terminology "function" refers to the *intended* activities of a device whereas "behavior" refers to its overall activities (intended and unintended). A function-preserving transformation is hence defined as one that preserves the functionality or design intent of the original specifications. A re-organized version of the graph in Figure 2-6 is shown in Figure 2-7 after being altered by such a transformation.

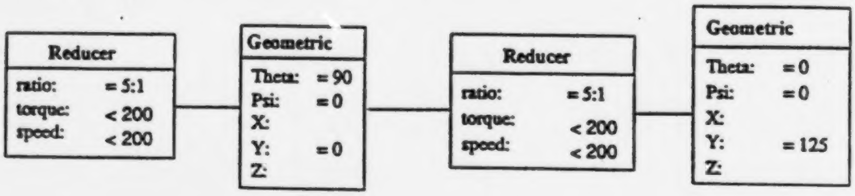

Reducer
ratio: = 5:1
torque: < 200
speed: < 200
Geometric
Theta: = 90
Psi: = 0
X:
Y: = 0
Z:
Reducer
ratio: = 5:1
torque: < 200
speed: < 200
Geometric
Theta: = 0
Psi: = 0
X:
Y: = 125
Z:

Figure 2-7: The transformed specification graph of a spur-gear drive (after Hoover and Rinderle 1989)

The synthesis strategy of Hoover and Rinderle (Figure 2-8) can be summarized as the following steps (from Hoover and Rinderle 1989).

1) Develop a specification graph that represents the functionality of the device to be designed.
2) Utilizing general engineering principles or domain-specific knowledge, apply any relevant general transformations to the specifications graph.
3) Of all the matching components (those with a form-behavior graph that matches part of the device's specification graph), find the one with the highest degree of functional integration as measured by the number of primitives matched in the specification graph.
4) If necessary, perform a validity transformation, i.e. re-organize the specification graph so that the elements of the resulting graph map onto the valid ranges of some existing components.
5) Instantiate the best component (found in previous step), that is, solve its design equations to get a parametrically-complete description of the component.
6) Apply an instantiation transformation to the specification graph to reflect the functional and physical characteristics of the selected component.
7) Repeat the above process until a complete design configuration is generated.

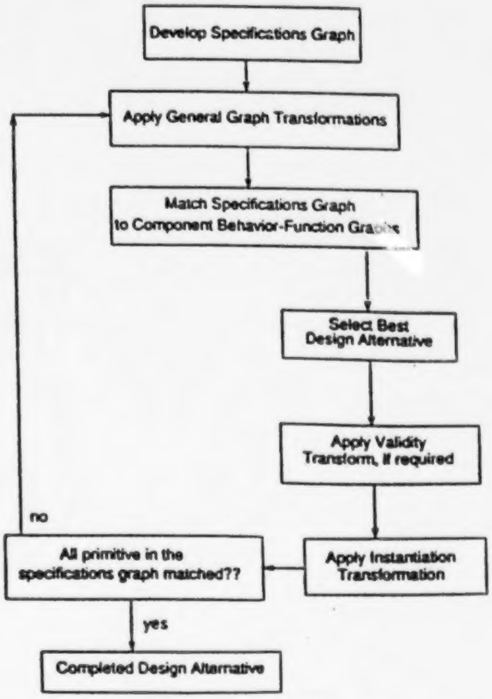

Develop Specifications Graph
Apply General Graph Transformations
Match Specifications Graph to Component Behavior-Function Graphs
Select Best Design Alternative
Apply Validity Transform, if required
Apply Instantiation Transformation
All primitive in the specifications graph matched??
no
yes
Completed Design Alternative

Figure 2-8: Functioning of the *synthesis strategy* (after Hoover and Rinderle 1989)

Hoover and Rinderle report that steps 3 through 7 of this methodology have been successfully applied to the domain of single-speed mechanical power transmissions and that a program has been developed to partially implement these steps. The following observations are true of the above synthesis strategy.

- The strategy has been implemented and tested in a very limited domain, namely that of the spur-gear design, and no attempts have been reported towards its generalization. It requires that the validity- and instantiation transformations be separately defined for each type of component. The same is true about the other procedures involved in the synthesis process. Devising techniques that are general enough to apply to a broad class of mechanical systems poses serious challenges to a methodology of this kind.

- Also, the strategy partially relies on decisions made by the designer/user, especially in steps 1 and 2 above. This is in contrast with the notion of design automation presumably intended in the development of the strategy. Special problem-solving and programming techniques are required to reduce the dependence of the methodology on the user's decision-making abilities.
- The methodology aims at generating one (presumably the best) design solution to a problem. It uses a best-first search strategy whereby in each stage of the design process it determines and selects the best component available. As the authors admit, however, this strategy does not necessarily lead to the desired outcome. This is because in a search tree, the path representing the globally optimal solution does not always comprise an elite set of internal points. It is quite possible that by rejecting a sub-optimal internal node one will lose the winning path. This issue will be further discussed in the next chapter.
- The authors point out that the overall functionality of a device cannot be represented by just an unorganised set of (form and function) primitives and that their configuration and interrelations have to be taken into account as well. Yet the reported strategy fails to make provisions for checking these in a design. According to the strategy, a component *will* be selected if and only if its schematic description matches any part of the initial specifications, regardless of whether or not the precedence order of the components and their interrelationship are fulfilled. This issue will be further discussed in the following chapters.
- The strategy uses a separate set of case-specific primitives to develop the schematic descriptions. This means that each system/component is presented in a vocabulary of its own. Communications among various parts of a design as well as between various designs rests upon the existence of a universal vocabulary of standard functions. Without a common language, it will not be possible to process a design in its entirety and to assess its performance.
- The design problem is considered "static" by the proposed strategy, that is, it is assumed that the specifications and requirements of the problem remain the same throughout the process. This is not true of real-world problems where each component of a system, once selected and instantiated (parametrically designed), may pose new constraints on the entire system, especially its adjacent components. Hence the least a design system should do is to provide means for intermittently checking the candidate design(s) in the

light of the emerging system-level constraints.

In the following chapters we shall discuss a number of methods to fix the above problems.

The concept of "dynamic" design problems, as opposed to static ones just mentioned, has been well explored by Serrano and Gossard (1987; 1988). They assert that much of the engineering design process involves the recognition, formulation and satisfaction of constraints and that constraints are continually being added, deleted and modified throughout the development of a device. Based on this belief, they have developed a conceptual design system, called the *concept modeller,* that allows the user to "conceptually experiment with various design alternatives" although it does not perform any design synthesis activity of its own. In other words, using the concept modeller, a designer can interactively construct conceptual models of his/her design ideas and analyze them to learn about their feasibility.

Two main metaphors form the basis of the concept modeller: the *building block* metaphor and the *dependency-sphere* metaphor. The former provides the user with the tools to construct a conceptual model of their design by assembling standard building blocks (icons representing mechanical elements) whereas the latter gives them necessary tools (basically constraint-management techniques) to study the performance and feasibility of the generated designs.

The system architecture of the concept modeller is shown in Figure 2-9. The *concept base* consists of a library of component-icons. Associated with each icon are the component properties describing its physical nature (form, size, material, etc.) and its behavior (physical laws, etc.). The default values of component properties can be overwritten by the user. The *working memory* is the work bench on which the icons are manipulated and design concepts are constructed. It displays the status of the design and the values of the design parameters at each time. The user can interactively make changes and additions to this memory. The *constraint manager* acts as the inference engine for the concept modeller. It has four basic jobs: constraint evaluation, minimization of computations, consistency maintenance and qualitative reasoning.

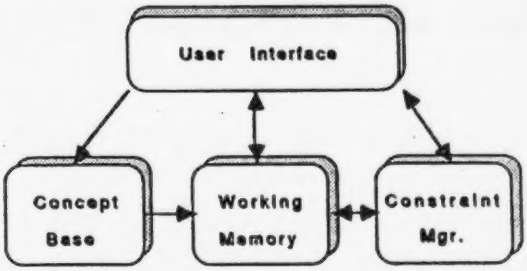


Figure 2-9: The system architecture of the "concept modeller"
(after Serrano and Gossard 1988)

Once an icon (component) is picked by the user and posted on the working memory, the constraint manager comes into play and does the following.

- Evaluate the governing constraints of the selected component for given values to reveal the order in which the constraints should be solved and the parameter which can be calculated by each constraint.
- Identify and isolate those constraints relevant to a requested computation through tracing dependencies among parameters.
- Spot the conflicts and inconsistencies among the constraints and bring them along with their sources to the attention of the user. The user then resolves the conflicts by changing the values of some parameters or their assignments to constraints.
- Determine the sensitivity information among the constraints, that is, which "strings" should be pulled (which input parameters should be changed) to result in a desired change in certain output variables.

In order to do all this, the concept modeller builds and uses a graphical network to represent the constraints and their relations. A constraint network is basically a directed graph in which the nodes represent design parameters and the arcs represent the constraint relationships. Let us illustrate this by an example problem of a welded cantilever beam (Figure 2-10(a)). The problem constraints and corresponding constraint network are presented in Figures 2-10(b) and 2-10(c) respectively. An evaluation of the constraint set results in the assignment graph presented in Figure 2-10(d). It gives the primary and secondary assignments of parameters to constraints, recommending which parameter be calculated from which constraint. This in turn allows the constraint network of Figure 2-10(c) to be converted to the tree-like structure of Figure 2-10(e).

Now that the converted constraint network is in the form of a simple tree, one can simply move up the tree from the leaf-nodes and calculate the unknowns. In this case the path would be as follows.

- Substitute the values of h, b and d in f1 (Figure 2-10(b)) and calculate I.
- Substitute the same value of d in f3 and calculate c.
- Substitute the values of F and l in f4 and calculate M.
- Substitute the calculated values of M, c and I in f2 and calculate σ.

The tree also shows which input parameters should be touched to result in a change in a desired output parameter.

The methodology of Serrano and Gossard offers valuable insights into the complex process of engineering design especially from a constraint-management perspective. It takes into consideration the dynamic nature of the process and, with this in mind, provides the designer with the necessary tools to construct, manipulate and evaluate conceptual models

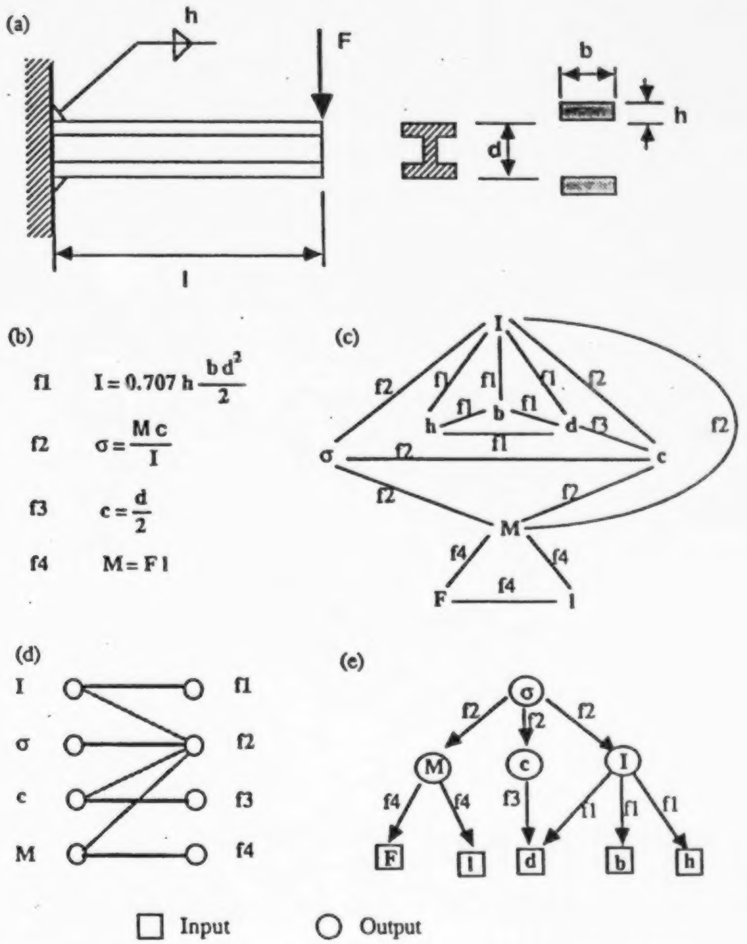

(a)
h
F
b
h
d
l
(b)
f1 I = 0.707 h bd²/2
f2 σ = Mc/I
f3 c = d/2
f4 M = F l
(c)
I
σ
h
b
d
c
M
F
l
f1
f2
f3
f4
(d)
I
σ
c
M
f1
f2
f3
f4
(e)
σ
M
c
I
F
l
d
b
h
f1
f2
f3
f4
Input
Output

Figure 2-10: A welded cantilever beam example and its solution steps
(after Serrano and Gossard 1988)

of mechanical systems. However, the methodology cannot be, and is not intended to be, used as a stand-alone conceptual design system because a computer system based on this methodology relies heavily on the interventions by the user. Also,

- the system bypasses the elements of "configuration" and "interactions" in design. It treats complex devices simply as "bags of components" where the configuration and precedence order of the components do not have any significance at all. It also ignores the interactions between various components of a design and assumes that the physical and functional presence of a component have no effect on the behavior of the others.
- the constraint-management techniques embedded in the methodology only consider the equality constraints and ignore the inequality constraints inevitably present in any real

engineering problem. As we shall discuss in the following chapters, many of the requirements of engineering design problems, both at system level and component level, are expressed as inequality constraints.

- the methodology involves a deterministic approach to constraint management. It idealistically requires that the equation set of a component be "exactly constrained" before it attempts a solution, meaning that there must always be exactly enough initial information to render an "**n** algebraic equations in **n** unknowns" case. This, however, does not conform with the reality of engineering practice where the designer has to put up with whatever initial information is available and where this often means an over- or underconstrained constraint set. Also, once the required information is provided, the system will find the same, single solution to the problem every time it is activated. Later in this report we shall introduce techniques that will enable the system to generate a broader variety of design alternatives both qualitatively and quantitatively.

Rao et al (Rao, Wang and Cha 1992) propose a different model for conceptual design of mechanical products. The Design-Analysis-Evaluation-Redesign (DAER) model emphasizes the last three stages (i.e. the analysis-evaluation-redesign cycle) while relaxing on the first one. Unlike those models that invest the most on "generating" a design based on the requirements, the DAER model requires that a preliminary design be given to it so that it can iteratively analyze it, evaluate it and, if not satisfactory, redesign it. According to the model, this preliminary design can be produced randomly, inspired by existing designs or generated by a human designer so that it meets all or part of the initial requirements.

Based on the DAER model, Rao et al have developed a design environment called Integrated Distributed Intelligent Design Environment (IDIDE) which can be used to develop special-purpose conceptual-design expert systems. The general problem-solving strategy of the IDIDE is presented in Figure 2-11 and is summarized below (from Rao, Wang and Cha 1992).

1) Define the problem for design tasks by "transforming the application environments and design purposes to functions", where the functions are taken from an "expertise function memory".
2) Transform the functional description developed in the previous step to a structural description using building blocks from a structure memory. If the existing building blocks are not sufficient, new ones have to be devised.
3) Develop the detailed description(s) of the design generated in the previous step using specific design knowledge stored in a "models memory" in the form of object-oriented frames (parametric design).
4) Analyze the product(s) of step 3 using numerical methods (e.g. statistical analysis,

optimization, etc.) from a "methods memory". Only the feasible designs are kept and the infeasible ones (not satisfying all requirements) are rejected.

5) Using techniques from fuzzy mathematics and system engineering, evaluate the feasible design(s) of step 4 and find the optimal one.

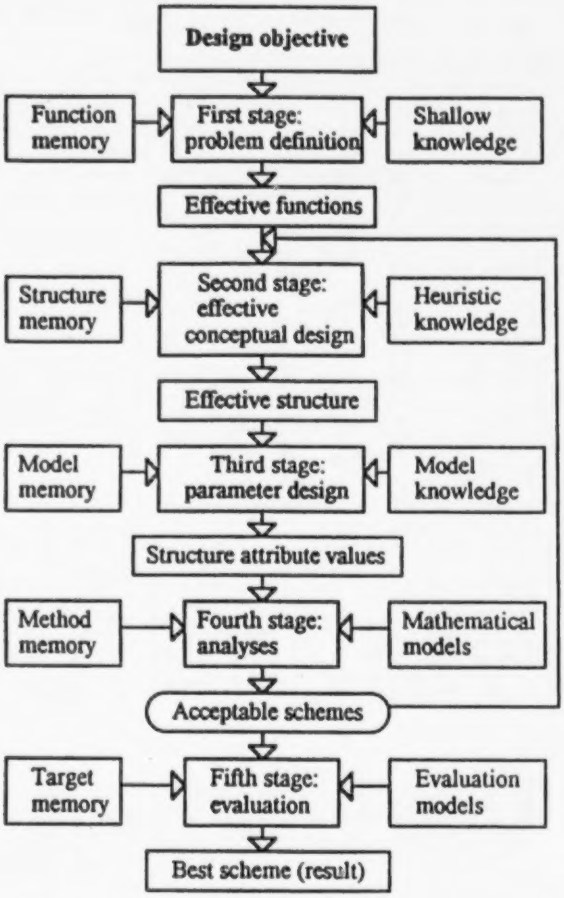

Design objective
Function memory
First stage: problem definition
Shallow knowledge
Effective functions
Structure memory
Second stage: effective conceptual design
Heuristic knowledge
Effective structure
Model memory
Third stage: parameter design
Model knowledge
Structure attribute values
Method memory
Fourth stage: analyses
Mathematical models
Acceptable schemes
Target memory
Fifth stage: evaluation
Evaluation models
Best scheme (result)

Figure 2-11: General IDIDE problem-solving strategy
(after Rao, Wang and Cha 1992)

The proposed methodology provides necessary tools for developing expert systems capable of designing individual mechanical systems and is especially good at reasoning and decision-making aspects of design. However, the details of the five implementation steps just outlined, especially the first two, have not been reported. It is not clear how the functional design (step 1) and conceptual design (step 2) are practically carried out and which computational techniques have been exploited. The complexity of these two steps has thus far impeded the full automation of mechanical design and the immaturity of existing computational and reasoning techniques has forced researchers to focus on limited portions of the process or limited classes of mechanical systems. A complete evaluation of the methodology therefore is not possible at this time.

Also, according to our definition of conceptual design in Chapter 1, "generating an abstract description of a device in terms of component-icons", which here is partially left to the user, is the whole purpose of conceptual design. The rest of the process (analysis, evaluation and redesign) is what some researchers refer to as *design refinement* and is generally not considered part of the conceptual design process.

A *function logic* approach to conceptual design has been proposed by Sturges (1990; 1992). It uses a set of linguistic and hierarchic rules to develop and analyze function-block-diagram representations of mechanical devices. In the work of Sturges, a FBD has three types of information associated with it: *function blocks* (compact verb-noun descriptors of what the design does), *allocations* (constraints, performance requirements, specifications and resources) and physical *components*.

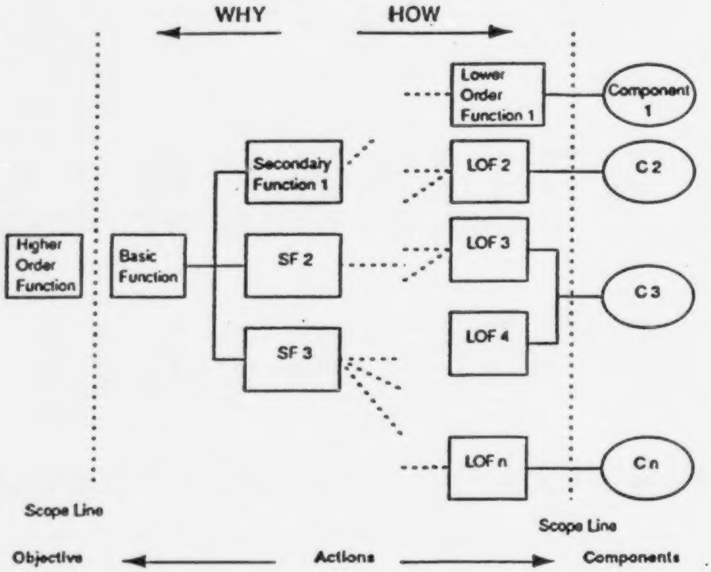

WHY
HOW
Lower Order Function 1
Component 1
Secondary Function 1
LOF 2
C 2
Higher Order Function
Basic Function
SF 2
LOF 3
C 3
LOF 4
SF 3
LOF n
Cn
Scope Line
Scope Line
Objective
Actions
Components

Figure 2-12: General structure of a *Function Logic Diagram*
(after Sturges 1992)

The general structure of a FBD, in this case called a Function Logic Diagram, is shown in Figure 2-12. Higher order (i.e. more abstract) function(s), representing the design objective, are placed at the left end of the diagram. As one moves right-bound in the diagram, these abstract functions get decomposed to lower-level (i.e. more specific) functions and finally, at the right end of the diagram, are replaced by component-icons. This way, a completed diagram would embody the conceptual design process, that is, the transformation of an abstract functional description of a device to its structural description.

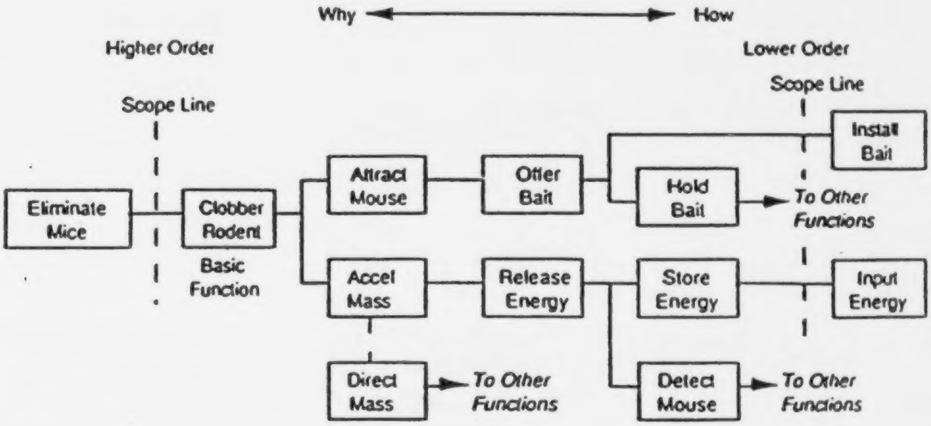

Why
How
Higher Order
Lower Order
Scope Line
Scope Line
Eliminate Mice
Clobber Rodent
Basic Function
Attract Mouse
Offer Bait
Install Bait
Hold Bait
To Other Functions
Accel Mass
Release Energy
Store Energy
Input Energy
Direct Mass
To Other Functions
Detect Mouse
To Other Functions

Figure 2-13: The function logic diagram of a mousetrap
(after Sturges 1992)

The function logic diagram of a mousetrap is partially shown in Figure 2-13 as an example. The rightmost function blocks (leaf-nodes) will ultimately be replaced by physical components. According to Sturges' paper (Sturges 1992), this replacement is carried out "by the designer before the detail design begins". It is not clear whether or not this implies direct intervention by the user, and in either case how the implications of multifunctionality and form-function relations in mechanical elements are handled. If the function of the proposed function-logic methodology goes just as far as function decomposition, then it will more properly fit the definition of "functional design" rather than "conceptual design".

Further evaluation of the function-logic model is not possible at this time since a problem-solving strategy and corresponding computational techniques have not been reported.

A number of other conceptual design systems have also been proposed/developed. Most of these systems are either custom-made to fit the requirements of certain industries/products or still at an early stage of development. In either case here we do not categorize them as general conceptual-design systems and will not discuss them in detail.

This is because many of the challenges mentioned in this chapter and elaborated in the following chapters do not arise unless one attempts to generalize the design technique to encompass a vast range of mechanical systems and gets to implement these techniques. It must be clearly stated what "design paradigm" a proposed design system is based on, which knowledge-sources are presumed available and what strategies are used to exploit these, usually diverse, knowledge-sources to get to the solution state. Otherwise, a comprehensive and realistic evaluation of these systems will normally not be possible. In the following paragraphs we shall nonetheless mention a number of these systems.

Motivated by a Ford Motor Company project to partially automate the preliminary design of its cars, Colton et al. (1990), Colton and Ouellette (1993) and Ouellette (1992) have developed a Vehicle Related Object Oriented Model (VROOM) for the conceptual design of automobiles. The model adopts a direct functions-to-components-mapping strategy whereby specific functions of a car are mapped onto auto parts in order to generate a structural description of an automobile. Two commercially available packages, the parametric design system ICAD (ICAD User's Manual) and the blackboard environment GBB (GBB: User's Guide), have been used to implement the proposed design system.

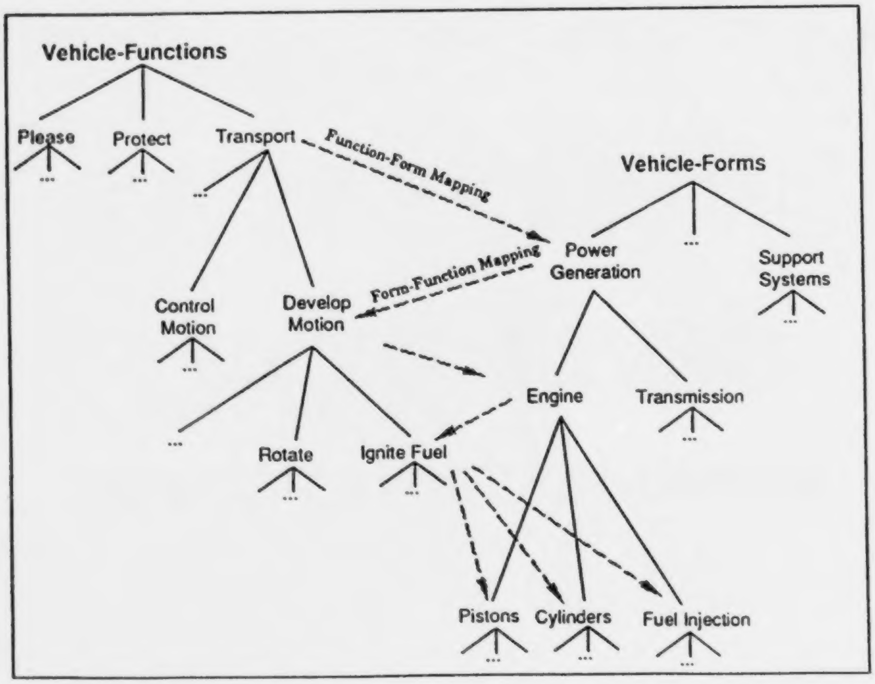

Vehicle-Functions
Please
Protect
Transport
Function-Form Mapping
Vehicle-Forms
Power Generation
Support Systems
Form-Function Mapping
Control Motion
Develop Motion
Engine
Transmission
Rotate
Ignite Fuel
Pistons
Cylinders
Fuel Injection

Figure 2-14: A typical function-form mapping of an automobile, based on the VROOM model (after Colton et al. 1990)

The vehicle is first decomposed into function and form hierarchies. Each of the three main functions (namely "transport", "protect" and "please") and three forms (namely "power generation", "chassis" and "body") of a car are decomposed into three-level hierarchies. For example, the function "transport" is decomposed into, among others, control motion (2nd level) and "steer vehicle" or "accelerate vehicle" (3rd level). These 3rd-level functions are then somehow mapped onto the same-level forms (components/subsystems). A typical mapping of this form is illustrated in Figure 2-14. Again, the mechanism of this mapping and the way the secondary functions of mechanical components are taken into consideration, if at all, have not been discussed.

Zhang and Rice (Zhang and Rice 1992) assert that "the essential feature of a mechanical system is that there is motion involved" and that any mechanical system can be conceptually represented in terms of seven "functional blocks": working block, driving block, transmission block, control block, support block, tribological block and auxiliary block. Based on these assertions, they propose a methodology for developing expert systems capable of conceptual design of "standard" devices. A "standard" device is defined as one that is "well specified by codes, standards or corporate procedures".

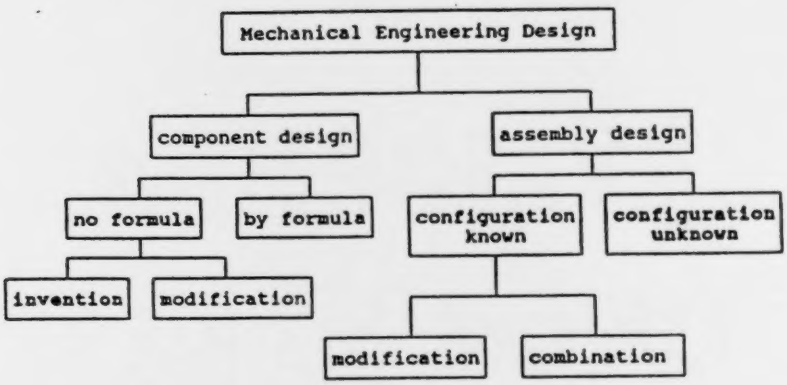


Figure 2-15: Classification of mechanical design problems according to Zhang and Rice (after Zhang and Rice 1989)

In their methodology, knowledge of different devices and their functions is stored in the form of "information chunks" with their names known as "tokens". Each chunk contains information on function, properties and detailed design data (e.g. dimensions, materials, manufacturing and maintenance requirements) of a device. Another group of tokens are the "configuration" tokens representing pre-defined abstractions of topological configurations.

During the design process, the initial "need" is matched against the "functional slot" of configuration tokens to select proper configurations. Then, guided by this configuration, design primitives (functional and consequently physical) are selected. If more than one design alternative is found, the "advantages and disadvantages" of these are deduced and compared, and the best is selected.

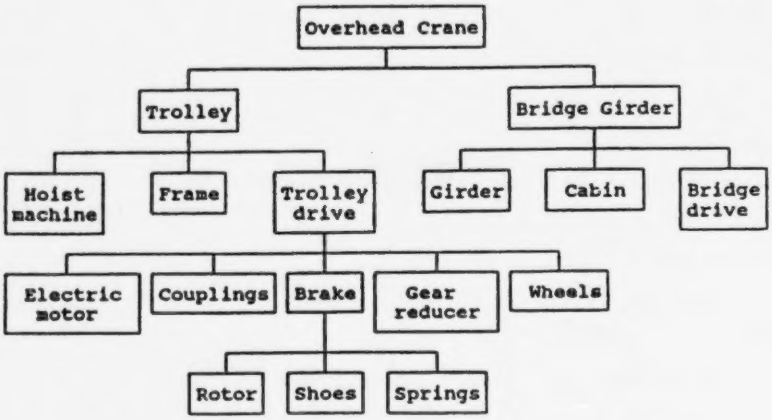


Figure 2-16: Pre-defined configuration of an overhead-crane
(after Zhang and Rice 1989)

The proposed methodology has been implemented in the context of Overhead-Crane design. According to Zhang and Rice's classification of mechanical engineering design problems (Figure 2-15), the problem of crane design is of the "assembly design" type. Furthermore, they acknowledge that the configuration of the device is known in advance (Figure 2-16) and that the job of the design system is to answer such questions as "is an auxiliary hoist needed?", "how many wheels are to be used for trolley drive?" and "what type of driving device (AC or DC motor) should be used?" This deviates from our definition of conceptual design in Chapter 1 where we included in the job description of a conceptual design system the ability to decide the components of a device and their configuration based on a functional description of the device.

The crane expert system uses Object-Oriented Programming techniques to model the crane component-types and component-instances as "classes" and "instances" and thus maintains its flexibility to deal with a variety of crane types and specifications. However, because the components are treated as individual elements, the system fails to consider the interactions among various components of the crane. For example, the effect of choosing 6 wheels instead of 4 on the natural frequency of the crane, or their weight on the design of axles are not taken into account.

Based on the theory of "Relational Database" (Codd 1970), Lai (1987a; 1987b) and Lai and Wilson (1987) have developed an automated system for conceptual analysis of mechanical products. The rational behind the development of the system is that "most new mechanical designs in industry are based on ideas from previous designs", and that with a "functional rationalization" of an existing design, it is possible to improve the quality of it and to generate new, better designs. The proposed design methodology comprises the following steps.

- Decompose a primary design into modules and components,
- Describe the architecture of modules, the assembly relations between components and modules in terms of functions,
- Identify the redundant functions, overlapped functions and mis-assigned functions,
- Improve the design by eliminating redundancy, rearranging remained functions, and by introducing new ideas.

Lai's design methodology succeeds somewhat in taking into consideration the significance of functional redundancies and interactions in design, but then again it fails to go past the functional reasoning barrier and take into account the implications of the design's physical existence. Hence physical aspects of mechanical devices such as geometry, material and dynamical characteristics do not play a role in the evaluation of a design. Also, as mentioned earlier, the function of the proposed methodology falls in the "design refinement" category rather than conceptual design.

With this we conclude our review of the related work. Some of the works reported here will be revisited later for comparison purposes, and many of the concepts introduced in the course of our discussions will be elaborated in the following chapters.

CHAPTER 3

A COMPUTER-BASED MODEL OF MECHANICAL CONCEPTUAL DESIGN

3.1 THE BASIC IDEA

As argued earlier, Mechanical Conceptual Design cannot be adequately represented by a direct, one-to-one mapping of the set of desired functions onto a set of known physical elements. This is because

I) a single element may contribute to more than one desired function and more than one element may be required to achieve any single function, and

II) the overall functional behavior of a design cannot be determined unless its form-function relations at both system and component-level as well as the interactions among its components are taken into consideration.

Since *constraints* are an integral part of any representation of the form and function of mechanical devices, the two statements above imply that some degree of *constraint analysis* must be performed *during* conceptual design rather than being entirely left for a later parametric design phase. Without such an analysis, values of the design/performance parameters cannot be determined and therefore the knowledge about the various functions a component can perform will be limited, at the most, to a qualitative level. This, plus the fact that interactions among components of a system are tightly related to their structural/mechanical characteristics such as geometry, weight, natural frequency and carried load, will preclude an understanding of the overall functioning of the system and will likely result in poor or even infeasible designs.

Suppose, for example, that in a transmission system, power is to be transmitted from an input shaft to a parallel output shaft, and that a conceptual designer has decided to do this using a worm-gear set. From a generic description of the subsystem "worm-gear" we know that it can "transmit power", "change rotational speed" and "change axis of rotation". This would probably be as much information as we can get if we chose not to examine the "dossier" of the subsystem including its constraint set. Then we would not know the types (axial/radial) and magnitudes of forces applied to the shafts and hence would not be able to decide the bearing types (radial/radial-thrust bearings, ball/cylindrical-/spherical-/tapered roller, etc.) to support the shafts. Also, we would not be aware of the incidental function "locking" of the worm-gear and that we could possibly exploit this to spare a brake in systems such as elevators and automatic doors.

To solve the problem just described, two general strategies have been proposed. The first strategy (Ulrich and Seering 1988; 1989) prescribes a *functional debugging* following a direct substitution of physical elements for functional elements. The procedure is to systematically look for opportunities to simplify the raw design by using additional functions of the physical elements. In other words, the designer examines the raw design and finds and removes those redundant elements whose functions are covered by other elements as their secondary functions. This strategy has been sucessfully applied to the schematic synthesis of single-input single-output dynamic systems, as explained in Chapter 2.

The second strategy, mostly advocated (though not explicitly stated) by the fans of *systematic design* (see Crossley 1980 for example) suggests that the direct function-to-physical element mapping be followed by an evaluation stage and in case of an infeasible/poor design, the transformation be repeated with alternate mappings until a satisfactory design is achieved.

We assert that the most efficient strategy for solving the problem of design functional redundancy/discrepancy, as introduced above, is to prevent it from happening in the first place by incorporating an "instantiation" stage in conceptual design.

The notion of instantiation in conceptual design has been, directly or indirectly, mentioned in some of the recent works on design automation. Hoover and Rinderle (1989) define it as "creating a complete (at least parametrically complete) description of the designed component" by "using the form-behavior relations for a class of components". In a generalization of the idea, we define *instantiation* in the context of conceptual design as *processing the available knowledge about a component/subsystem to learn about its functional behavior as well as those physical aspects that affect its relations with other components.*

The "knowledge" mentioned in the last paragraph typically comprises the component-specific facts/rules/data from pre-packed *knowledge cells* plus the initial information/ requirements of the problem. Included in instantiation is the examination of the problem's constraint set to determine the values of the component's design/performance parameters. This whole idea will be further elaborated in this and the following chapters.

Based on the above discussions we present a computer-based model of mechanical conceptual design, called *Design by Exploration,* with an emphasis on design automation. Earlier in this work we argued that design automation is best achieved through the implementation of computer-based, rather than cognitive, models of design. Design by Exploration is one such model. It fundamentally relies on the use of problem-solving techniques specifically designed to be carried out on computers, although it occasionally prescribes methods that are as well used by human designers.

In order to complete the foregoing discussion, we introduce yet another classification of the design models proposed by Kannapan and Marshek (1992). They divide design models into two groups: those that regard design as a natural process and those that regard it as an artificial one. The former implies that "design methodology is based on creativity and heuristic knowledge and hence in the realm of natural human behavior", whereas the latter imply that the methodology is "formal and mathematical and hence in the realm of symbolic data representation/manipulation".

According to our definition of various design paradigms (Chapter 1) and considering the above classification, a computer-based model would represent an artificial process and should therefore embody one of the seven approaches mentioned by Kannapan and Marshek, namely "System Science", "Problem Solving/Planning", "Transformational", "Database", "Algorithmic", "Axiomatic" and "Machine Learning".

Although Design by Exploration does not exactly fit into the definition of any of the above, it can be best categorized as a *transformational* approach. A transformational approach is formally defined (Kannapan and Marshek 1992) as "a sequence of correctness-preserving transformations of the initial design requirements to a final design description sufficient for implementation/manufacturing". In other words, the initial functional description of an artifact is successively refined/reconfigured (without altering the overall behavior of the artifact) until a representation is obtained which corresponds closely to a collection of existing physical elements. This collection will then represent the desired design generically, and is to be parametrically designed before it can be manufactured.

The following section is devoted to a detailed presentation of the proposed model and to explain the ways in which it differs from conventional *transformational* paradigm.

3.2 DESIGN BY EXPLORATION (DbE)

DbE builds on the assertion that feasible/optimal designs may be obtained through a stepwise transformation of a functional description of a device to a structural description provided that

a) at each step, the *knowledge pool*[1] of the problem is examined and the behavior-defining parameters are determined (instantiation) so that the overall functional behavior of each component/subsystem, its interactions with other components and its contribution to the satisfaction of initial functional requirements can be explored, and

---

[1] The notion of *knowledge pool* will be further discussed in the rest of this chapter.

b) throughout the design process, all *feasible* (with respect to various lifecycle requirements) partial design alternatives are nurtured and preserved for a final choice-stage where the user can pick the optimum alternative(s) according to their own evaluation criteria. This is to avoid the possible loss of the optimum design due to a best-first search strategy, as the best internal node in a search tree does not necessarily lead to the best overall solution.

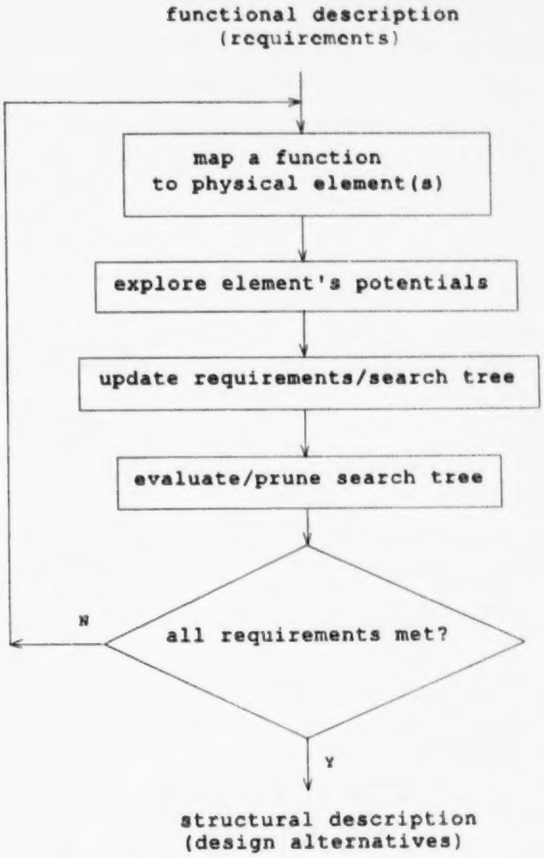

functional description
(requirements)
map a function
to physical element(s)
explore element's potentials
update requirements/search tree
evaluate/prune search tree
N
all requirements met?
Y
structural description
(design alternatives)

Figure 3-1: The *Design by Exploration* model

The Design by Exploration model is schematically shown in Figure 3-1. The underlying methodology can be outlined as follows.

1) Get the functional description of the artifact,
2) Map an unmapped function to as many existing physical elements/ subsystems as possible,
3) Explore the *knowledge pool* of each element to reveal its complete functional behavior,
4) Update the functional requirements,
5) Update the search tree by adding/ augmenting leaf-node partial designs,
6) Verify partial designs and prune the search tree by removing the invalid designs,
7) For each design alternative, if all initial functional requirements are met then quit, otherwise go to step 2.

We shall now explain these steps in more detail and illustrate them using a simple example. The example will be to conceptually design a machine that "rotates a cutting tool at certain speed and with certain power" (a drill). Let us add to this the requirement that the primary driving shaft (e.g. that of the electric motor) should be, for ergonomic reasons, perpendicular to the output shaft (convenience of use)[2]. For the sake of brevity, we shall only consider the main elemental functions and skip the computations.

3.2.1 THE INITIAL REQUIREMENTS

The problem is presented to the *conceptual designer* presumably in the form of a Function Block Diagram (Scetion 1.9) which is composed of Standard Elemental Functions. This presumption plays an important role in the implementation of the DbE model because, as we shall explain later on, it provides us with a common vocabulary (SEFs) in which both the functional requirements and the functions of physical elements can be expressed and hence the communication between the two sets would be possible.

We also remember from Chapter 1 that associated with any Function Block Diagram are the given specifiers plus the additional requirements/information not contained in the diagram. The latter is mostly expressed as a set of constraints in terms of the design/performance parameters of the problem. From now on, we shall refer to these constraints as the *External Constraints* to distinguish them from the *Internal Constraints* and *System Constraints* that are introduced internally, i.e. are either generated during the design process or implied by the system's knowledge-sources, as we shall explain later on.

As an example, let us consider the simplified requirements FBD of our example drill (Figure 3-2). As expected, the diagram is composed of standard elemental functions (Section 1.9.1). In terms of these basic functions the desired device should

- generate rotational motion,
- adjust the speed (and torque),
- redirect the axis of rotation in the desired direction (90°), and
- transfer the torque to the cutting tool (drill bit).

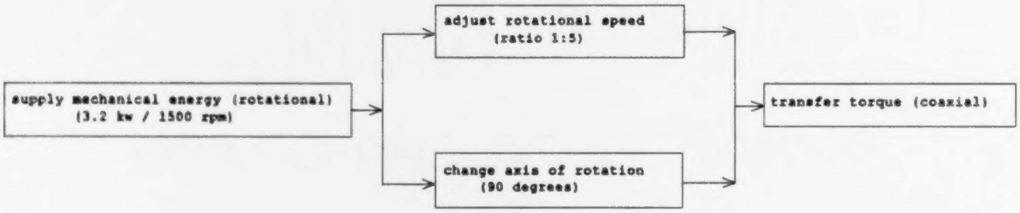

adjust rotational speed (ratio 1:5)
supply mechanical energy (rotational) (3.2 kw / 1500 rpm)
transfer torque (coaxial)
change axis of rotation (90 degrees)

Figure 3-2: The requirements FBD of a drill

---

[2] Note that the example is presented here for illustration purposes only, and neither the problem itself nor the solution to it are necessarily real and accurate. A more realistic example will be presented later in this work.

Note that we have not included all the numerical specifiers of the elemental functions since at this stage we intend to skip the computations as we work through the example. Handling of numerical specifiers will be treated in detail in Chapter 5.

The formal definition of the problem is about all the information the user is expected to provide to the system. The model envisions no other user interventions (e.g. decision making) during the course of design. However, as we shall see in the next chapter, the user will have the option of augmenting/modifying the constraint set of the problem during the design process, hence guiding the search in new directions.

3.2.2 FUNCTION-TO-PHYSICAL ELEMENT MAPPING

Standard Elemental Functions are, by definition, the smallest, non-decomposable building blocks of mechanical functions whereby both the functional requirements and the functions of the known mechanical elements are described. Therefore, if an element *can* perform a SEF (i.e. the SEF *is* included in the description of the element) then the element will definitely be detected and picked through a mapping from the functions set to the elements set.

The mapping is intended to reveal *all* the matching elements, that is, all the elements that potentially perform the desired SEF. This does not mean that all these elements will qualify as components of the design alternatives. Some of them will ultimately be rejected because of their unwanted secondary functions and/or because they are "quantitatively undesirable". This, however, will not happen until all these candidate elements are further "explored" and their entire functional behaviors are understood, as at the current stage no constraint processing takes place, even though the constraints concern the elemental function under consideration. So, for the time being, all the matching elements will be equally considered as candidates.

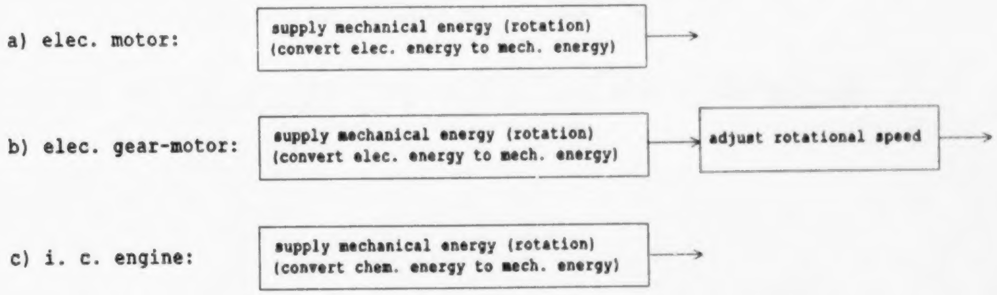

a) elec. motor:
supply mechanical energy (rotation)
(convert elec. energy to mech. energy)
b) elec. gear-motor:
supply mechanical energy (rotation)
(convert elec. energy to mech. energy)
adjust rotational speed
c) i. c. engine:
supply mechanical energy (rotation)
(convert chem. energy to mech. energy)

Figure 3-3: Three matching elements for the 1st elemental function

In the drill example, we start out with the first SEF (generating rotational motion) and search our database of *element-cells*. Figure 3-3 shows part of the findings of the search. These include an electric motor, an electric gear-motor and an internal combustion engine. As the figure shows, each of these matching elements/subsystems is also represented by a FBD. This is typical of all the elements in the elements database. This compatibility between the representations of functional requirements and that of the behavior of the elements is what makes the mapping process possible.

Having found all the matching elements, a search tree is now established with these elements as the first set of internal nodes. The leaf (terminal) nodes of this tree, when completed, will represent the final (complete) design alternatives whereas the internal nodes will represent the partial designs. As we shall discuss later on, a breadth-first strategy is employed to expand the tree through a series of augmentations/prunings.

3.2.3 EXPLORING THE ELEMENTS' POTENTIALS

The matching elements of the previous stage were chosen merely based on their prospect of performing one of the desired elemental functions. It is not yet known what other desired/ undesired functions they perform, whether they qualify for the job quantitatively and whether they violate any of the problem's constraints. To acquire this information, at this stage a *knowledge pool* is set up for each of the candidates (matching elements). The pool contains the following three general types of knowledge.

1- The element-specific knowledge from the pre-packed "element knowledge-cell". This includes the list of SEFs the element potentially performs, its internal constraints (design equations/charts/ graphs, catalogs), and any prominent behavioral characteristics not represented by these.

2- That part of the user-provided initial specifications (or more correctly the initial *knowledge*) that concerns the element at hand. This includes the external constraints (those imposed on the whole system as part of the initial requirements) and system constraints (those imposed by other components of the system, expressing the consistency requirements).

3- Other life-cycle (e.g. manufacture, cost, reliability) considerations introduced by special purpose expert modules and/or the user during the design process.

To clarify these knowledge types, let us consider a simple example. Suppose that a transmission system is to be designed in which a chain drive is proposed to reduce the rotational speed. The element-specific knowledge will then comprise the set of design equations of the element (internal constraints) and, in this case, another constraint representing the fact that a speed ratio of more than 6:1 is normally not recommended in chain drives (behavioral characteristic).

As another requirement, suppose that the whole transmission system is to fit in a limited space. This dictates that the centre distance of the two sprockets not exceed a certain length (external constraint). Also suppose that a pair of ball-bearings is to support each sprocket shaft and the bearings require that the diameter of the shaft, and consequently the hub diameter of the sprocket, lie within a certain interval (system constraint).

As for the "other life-cycle considerations", we further suppose that our financial concerns make cast iron the optimum choice of material for the sprockets. This in turn will restrain the allowable stress and will impose yet another constraint to be satisfied by the design. The fact that new constraints may be introduced by various knowledge sources *during* the design process reflects one of the most important aspects of real-world design problems. Regrettably, this "dynamic" nature of mechanical design has been overlooked by many of the design models proposed to date (see Chapter 2). These models typically assume that the entire set of problem requirements and the underlying constraints, given at the outset of the design process, will remain the same throughout the process. One implication of this assumption is the significant relaxation of the computational/reasoning requirements, as we would now be dealing with a static, rather than dynamic problem. We shall further discuss this important issue in subsequent sections.

Having set up the knowledge pool of the candidate element with its three knowledge types just explained, now the pool is to be processed to yield the information required for understanding the entire behavior of the element. "Processing" of the pool involves, for the most part, evaluating/solving the problem's constraint set to find the feasible values of the element's behavior-defining parameters. The outcome will be in one of the following forms.

- A number (≥ 1) of *feasible* sets of parameter values. Here the term *feasible* means that the set(s) of calculated values satisfy the *entire* constraint set of the problem, including the initial specifications of some parameter values. Each set will then represent an *element-instance*.
- A number (≥ 1) of sets of parameter values that satisfy all but a few of the problem's constraints, where the unsatisfied constraints represent a subset of the initial parameter specifications. In other words, the design system figures (we shall explain how) that no element-instances (and hence no solutions to the problem) can be found unless some of the given values for design/performance parameters are changed. Having applied those modifications, the constraint set is then solved for the rest of the parameters. As we shall discuss later on, this little "breath" of intelligence will enable the design system to make suggestions to the user, who will ultimately decide whether to accept or to reject the suggested solution(s).

- A message indicating that no solution for the given problem could be found, not even by sacrificing a number of initial specifications.

In the rest of this chapter we shall refer to the three forms above as *first-, second- and third-form outcome* respectively.

Back to the drill example, let us now apply the above to our three candidate elements, namely the electric motor, the electric gear-motor and the internal combustion engine (Figure 3-3). For the electric motor, the knowledge pool simply contains the initial specifications (power and speed) plus the contents of the "elec. motor knowledge cell" from a preloaded knowledge base. In this particular case, the latter contains a "catalog" whereby a (number of) motor(s) can be found for each power/speed entry (say, 3.2 kw and 1500 rpm). Each of these motors will be referred to as an electric-motor *instance*. The catalog will also provide some information about the other physical/functional characteristics of the motor, such as its dimensions, weight, heat generation and efficiency. However, in this simplified example we shall ignore that information as it does not contribute to the satisfaction of the requirements.

Note that the element-cell in this case does not report any other elemental function for the element. Also note that the outcome of the knowledge-processing is of the first form, that is, a number of feasible instances of the candidate element are found that satisfy all the given requirements.

Going on to the next candidate, the electric gear-motor, we get a knowledge pool similar to that of the electric motor. This time, however, a processing of the pool reveals a second SEF of the element, namely "adjust rotational speed" (Figure 3-3). Now suppose that the catalog tells us 3.2 kw gear-motors are only available in 2500, 1800, 1200 and 900 rpm, i. e. the gear mechanisms mounted on the motor shafts are only available with ratios of 5:3, 6:5, 4:5 and 3:5 respectively. As we see, none of these speeds/ratios is what we are looking for (1500 rpm * 1:5 = 300 rpm) (Figure 3-2). We therefore pick the closest choice (900 rpm or 3:5 ratio). This is a typical example of the second-form outcome (of the knowledge-pool processing), as all but one of the initial specifications have been sustained. In the next subsection we shall see how this situation is handled by the design system.

The third and the last candidate element at this stage is the I. C. engine. Again we suppose that a catalog search would give us what we want, i.e. an engine with 3.2 kw of power and the speed of 1500 rpm (another first form outcome).

One should note that at this stage we did not encounter a third-form knowledge-processing outcome where the design system concludes that a solution cannot be found for the problem at hand.

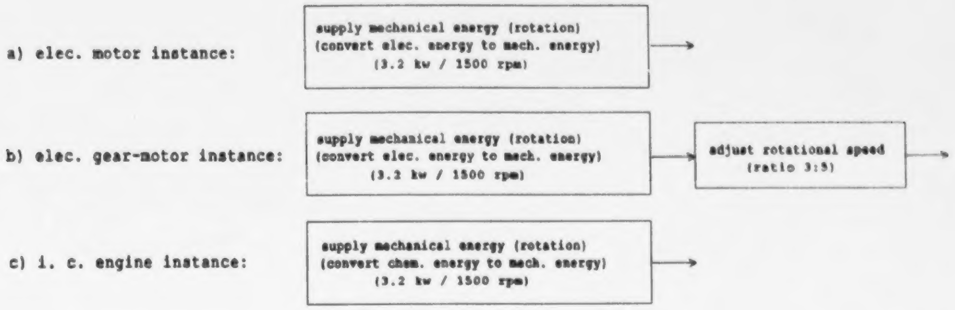

a) elec. motor instance:
supply mechanical energy (rotation)
(convert elec. energy to mech. energy)
(3.2 kw / 1500 rpm)
b) elec. gear-motor instance:
supply mechanical energy (rotation)
(convert elec. energy to mech. energy)
(3.2 kw / 1500 rpm)
adjust rotational speed
(ratio 3:5)
c) i. c. engine instance:
supply mechanical energy (rotation)
(convert chem. energy to mech. energy)
(3.2 kw / 1500 rpm)

Figure 3-4: Instances of the three candidate elements

Figure 3-4 shows the results of the *exploration* stage for the three candidate elements selected for their promise of performing the first required elemental function, i.e. "supply mechanical energy". The next step would be then to evaluate the contribution of these elements to the requirements of the problem.

3.2.4 UPDATING THE FUNCTIONAL REQUIREMENTS

At this point the candidate elements have been "explored" and some/all of them have been successfully instantiated (i.e. their entire functional and physical characteristics have been disclosed). This information tells us

- to which required elemental function(s) does each element-instance contribute, and
- whether these contributions are quantitatively adequate.

In other words, based on the outcome of our exploration we can tell a) which SEFs are present both in the initial requirements and the functional behavior of the element instances, and b) if the element instances can perform the desired SEFs to the desired specifications.

In the drill example, for instance, we realized that both the electric motor and the I. C. engine can perform the first desired function (supply mechanical energy) to the desired specifications (3.2 kw at 1500 rpm), whereas the selected electric gear-motor performs one function (supply energy) wholly (i.e. to the desired specifications) and a second function (adjust speed) partially (to a ratio of 3:5 instead of 1:5). The situation implies that we would need other component(s) to complete the coverage of the partially-covered second elemental function (in this case perhaps another gear-set as we shall find out). In any case, i.e. whether elemental functions are completely or partially covered by selected elements, we need to find the answer to the question "what is left to be done to satisfy all the requirements?". The answer would naturally be one of the following.

- Nothing; the current arrangement(s) of physical elements meets all the requirements already, or
- Find new elements to satisfy the remaining, intact functional requirements, or
- Find new elements to satisfy the remaining, intact and/or partially satisfied functional requirements.

It is the purpose of this stage to find the answer to the question above, and to update the formal representation of the initial requirements accordingly. The latter is necessary because in taking any further actions, the design system depends on this representation.

Since both the initial requirements and the functional behavior of the selected element instances are expressed as Function Block Diagrams, the task at this stage could be carried out by comparing the two FBDs for each element instance. In doing so, the following steps are made for each element instance.

- Lists of elemental functions from both sides (the element and the requirements) are compared to spot the common SEFs,
- Specifiers (chapter 1) of the common SEFs from both sides are compared to see if they match,
- Those common SEFs whose entire set of specifiers match are marked "done",
- If a complete match does not exist for some of the common SEFs, their specifiers are adjusted to reflect the remaining demand, that is, the new specifications for the partially-covered SEFs are determined,
- Any accidental SEFs of selected elements, not already appearing in the requirements, are marked "extra" and added to the FBD representation of the requirements.

When the above is done for all the selected element instances, the modified FBD representation(s) of the requirements will be treated the same way as the original one in the subsequent iterations.

As for the "extra" functions, i.e. the ones inherently performed by the selected elements but not part of the initial requirements, there are two possibilities. Either the function is "harmless" to the design and does not impose any unwanted characteristics to the design or it does. An example of the former is to use a conical roller bearing to support a shaft with only radial loads, where the bearing's other function "support axial load" is extra, but harmless to the design. On the contrary, if a bevel gear set is used to carry out the function "adjust rotational speed", it will also perform the extra function "change axis of rotation" which might not be generally desired. In either case we may face one of the following situations in the course of design.

- The extra function(s) are negated by other, upcoming elements (e.g. the unwanted right angle caused by the bevel gear set is "corrected" by another gear set of the same type).

Here no action needs to be taken to remove these functions.

- The extra function(s) are harmless and are not cancelled out by other functions. Again no actions are necessary in this regard.
- The extra function(s) do cause some unwanted effects and are not cancelled. In this case the partial design will be spotted and rejected either by the design system (for its constraint violation, as we shall see in the next subsection) or ultimately by the user at the end of the design process.

We also note that when it comes to adjusting the function specifiers to reflect the contribution of an element instance, we might face one of the two types of specifiers: the *adjustable* and the *unadjustable*. The adjustable specifiers are those that accept contributions from a number (≥2) of elements and whose "shortage" can be compensated for by adding other elements which perform the same function and use the same specifiers. For instance, if a spring stiffness of "k" is required in a design and it so happens that a selected spring can only provide a stiffness of k_1 ($k>k_1$) due to some design constraints, a second spring of stiffness k_2 can be added in parallel with the first one so that the overall stiffness would be equal to k (from the relation $k = k_1 + k_2$). We would therefore call the spring stiffness "adjustable". On the other hand, if a gear set was selected to transmit certain power at certain rotational speed and one or both gears could not be designed for that speed (due to some design constraints such as dynamic loading), we would not be able to compensate for the speed by, say, adding another gear set. Hence for this particular case we classify the rotational speed of the gear as unadjustable.

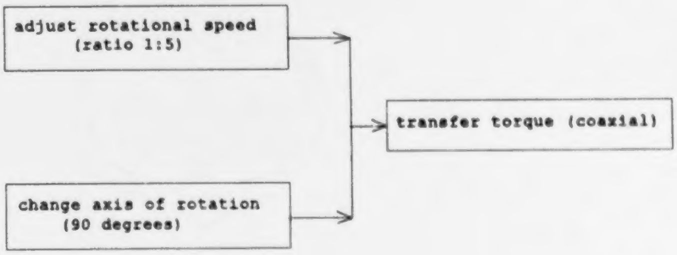

adjust rotational speed (ratio 1:5)
change axis of rotation (90 degrees)
transfer torque (coaxial)

Figure 3-5: Updated requirements for elec. motor and I. C. engine

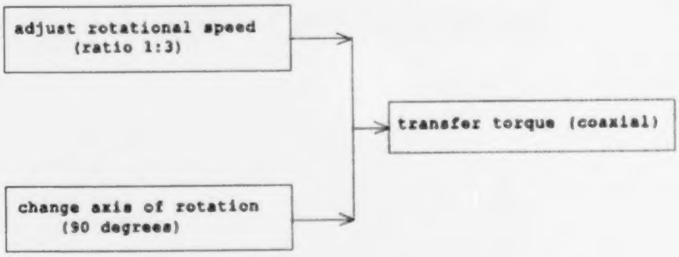

adjust rotational speed (ratio 1:3)
change axis of rotation (90 degrees)
transfer torque (coaxial)

Figure 3-6: Updated requirements for elec. gear-motor

Let us now go back to the drill example

and "update" its requirements. Comparing Figures 3-2 and 3-4, we realize that to update the FBD of the requirements (Figure 3-2) for the electric motor and the internal combustion engine only the first function block (supply mechanical energy) should be marked "done" and the rest of the diagram remains intact (Figure 3-5; note that for simplicity, we have only shown the updated diagram and removed the "done" blocks).

In case of the electric gear-motor, the first function block is marked "done" and the specifier of the second block is modified (new ratio "1/3" = required ratio "1/5" ÷ attained ratio "3/5") (Figure 3-6).

3.2.5 SETTING UP/UPDATING THE SEARCH TREE

As mentioned earlier, the Design-by-Exploration model seeks to generate multiple design alternatives rather than a single optimal design. It uses a breadth-first strategy to build a search tree in which the internal nodes represent partial designs and the leaf nodes represent complete, feasible alternatives. To better picture this, let us for a moment forget where we left the drill example and jump ahead a few iterations to the hypothetical situation illustrated in Figure 3-7. The figure shows part of the drill's final search tree in which the nodes have been numbered for reference. A path from node 1 to each of the leaf nodes (9, 10, 11, 12 and 13) denotes a complete design. For example, path (1-2-6-11) represents the arrangement of the motor, a pair of spur gears (to partially reduce the speed), a pair of bevel gears (to further reduce the speed and provide the required right angle) and a chuck. A total of five conceptual design alternatives are shown in this figure.

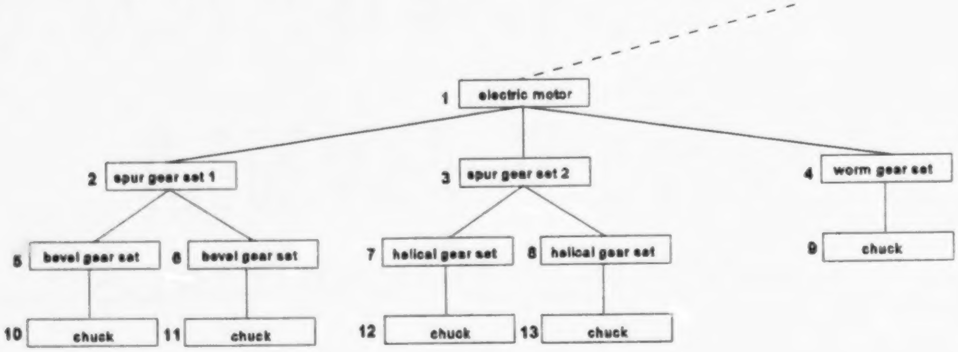


Figure 3-7: Partial search tree of the drill example

This "multiple, feasible-" (rather than single optimal-) design generation is one of the main features of the DbE model. For a search-based design system to be able to find the optimal design(s) for given requirements we assert that

- the system must consider *all* possible alternatives and leave the system-level optimization for the final stage where these alternatives have been generated[3], and
- generation of all alternatives can only be guaranteed if at each level of the search tree, *all* feasible instances of *all* competent elements are considered. For example, in Figure 3-7 we have shown two instances of the spur gear set (nodes 2 and 3) with, say, the same ratios but different number of teeth and different face widths and/or different materials.

The main reason for not considering only *one* (optimal) instance of *one* component at each level of the search tree is that *an optimal partial solution will not necessarily lead to the overall optimal solution.* It is quite possible that the globally optimal solution (say path 1-3-7-12 in Figure 3-7) is lost by discarding a sub-optimal internal node (say node 3) in the search tree. It is also possible that an optimal partial design would turn out to be infeasible because it would violate some of the constraints posed by subsequent nodes in the tree.

Suppose, for example, that the gear set of node 2 in Figure 3-7 is minimal in size and weight. There will naturally be a constraint on the maximum inside diameters of the two gear hubs due to strength requirements. Further suppose that the torque requirements of the bevel gear (node 10), to be mounted on the same shaft as the spur gear of node 2, dictate a minimum shaft diameter which exceeds the maximum allowable hub diameter of the spur gear. The two component instances are obviously incompatible and the (1-2-5-10) alternative will be discarded. As a result, we might end up with a globally sub-optimal design (1-2-6-11; if compatibility requirements are met) or none containing spur gears at all otherwise.

For the reason just discussed, we prefer to save *all* the element instances successfully generated at the previous stage. Once generated, each new instance is added to the search tree (as a leaf node) at the end of its parent branch.

Back to where we left the drill example, we now have one instance of each of the three candidate elements. Using these instances, we now set up the search tree of the drill (figure 3-8). Note that we do not yet call the instances *feasible*, as we do not yet know whether they will satisfy all of the problems's constraints.

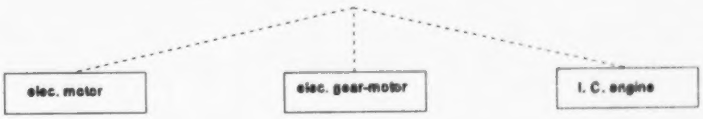


Figure 3-8: Search tree of the drill; first level

---

[3] Note that this assertion does not preclude the use of the expert knowledge of the domain to eliminate some infeasible parts of the search space at early stages, as this too must be preceeded by *generation* and *consideration* of the alternatives.

3.2.6 VERIFYING PARTIAL DESIGNS

In our discussion of the exploration/instantiation stage of the design process earlier in this chapter, we mentioned that the *knowledge pool* of a candidate element would contain *all* the constraints to be satisfied. This would obviously mean all the constraints given *at the time of exploration.* There are, however, some constraints that cannot be considered/evaluated at that time. These can be mainly divided into two groups: those constraints which are introduced to the system after the exploration stage has begun, and those which are already known to the system but whose evaluation rests on the results of the exploration.

Suppose, for instance, that after we have explored the candidate elements of the drill example, we are informed that the device will be used indoors as well as outdoors. This (qualitative) constraint fits into the definition of the first group. Also suppose that we have been told (from the beginning) that the drill's dimensions must not exceed certain values. This constraint falls into the second category as the dimensions of the motor/engine could not be looked up in respective catalogs until specific instance(s) of them have been selected; and hence the constraint could not be considered in the instantiation stage.

Another major class of constraints that pertains to the second category above is that of the FBD constraints. We remember, from the description of "function block diagrams" in chapter 1, that the *arrangement* of the blocks in a diagram generally represents their interrelations. Outputs of certain blocks may be inputs/control parameters to others and this "causality" requires that the sequence of function blocks be generally upheld. Nonetheless, in the course of selecting/exploring feasible elements we did not, and we could not readily, make provisions to take this into consideration.

To cope with this new problem (i.e. some constraints not being considered in the previous stage), the DbE model requires that the design system be equipped with a dynamic evaluation mechanism to let it continuously check all partial designs against the evolving set of constraints and discard the infeasible ones. This is crucial not only to guarantee the feasibility of the nodes of the search tree, but also to avoid a combinatorial explosion of partial designs. All partial designs are verified against the most updated version of the constraint set of the problem and the search tree gets pruned through the removal of the nodes not completely complying with the constraint set.

Back to the drill example, we now update the problem's constraint set by adding to it the two new constraints (indoor use and size limit). We then verify each of the partial designs (each having one element so far) against the updated constraint set. Let us suppose that we have looked up the dimensions of all three designs and that they all meet the size constraint. As for the indoor-use constraint, the electric motor and the electric gear-motor both satisfy the constraint whereas the I.C. engine fails due to its exhaust problem (example of a new functional requirement). The search tree then loses one of its branches and keeps the other two (Figure 3-9).

elec. motor
elec. gear-motor

Figure 3-9: Updated search tree of the drill; first level

As for the FBD constraints, a comparison between the FBDs of the two "surviving" partial designs (Figures 3-5 and 3-6) and that of the initial requirements (Figure 3-2) indicates that both designs comply with the precedence requirements of the latter.

As a counter-example, imagine that the system's database contained a mechanical element with the FBD shown in Figure 3-10. The selection mechanism of the system would pick it up and it would survive through all constraint checks. However, we note that according to the requirements diagram (Figure 3-2) the input to the "transfer torque" function block must be the output from the "adjust rotational speed" and "change axis of rotation" functions and not from the "supply mechanical energy" function. In other words, we do not wish to transfer a high-speed, low-torque rotation to a drill bit along the wrong axis, but we rather wish to adjust the speed/torque and the axis of rotation before transferring it to the bit. The FBD-constraints check at this stage are meant to spot and remove deceptive partial designs of this kind.

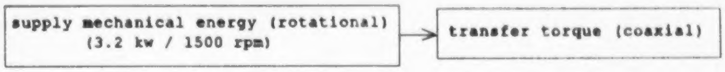

supply mechanical energy (rotational)
(3.2 kw / 1500 rpm)
transfer torque (coaxial)

Figure 3-10: FBD of a hypothetical element

3.2.7 ITERATING THE PROCESS

At the end of each iteration we get a pruned search tree with a custom-made requirements function-block-diagram associated with each leaf node. Each diagram tells us how successful the corresponding node (and the design alternative it represents) has been in satisfying the requirements of the problem. If the diagram of a node shows that no requirements are left unsatisfied, the corresponding node is marked "feasible design" and preserved for the final presentation of the results. Otherwise (i.e. the requirements FBD of a node reflects some uncovered functions) we loop back to the second stage, i.e. to the mapping stage, and repeat the process as if we were given a new problem.

Having elaborated various stages of the design process according to the DbE model, We now continue with the drill example briefly and show highlights of the search process for each partial design established so far.

STAGE 2: Starting with the electric motor and its requirements diagram (Figure 3-5), the system searches its elements database and finds six elements promising to carry out the first functional requirement (adjust rotational speed), viz.

- Spur gear set
- Bevel gear set
- Helical gear set
- Worm gear set
- Chain drive
- Belt drive

Similarly, for the electric gear-motor the system decides that the same six elements will work, except that this time around they will be required to perform some of the elemental functions with different specifier values (because in this design alternative, the function "adjust rotational speed" has been partially covered already).

STAGE 3: Exploring the knowledge-pools of the six candidate elements for both partial designs reveals that they all are capable of carrying out the first function to the respective desired extents (desired values of function specifiers, namely 1:5 for the electric-motor alternative and 1:3 for the electric-gear-motor alternative). Moreover, three of the above, namely the bevel gear set, the worm gear set and the belt drive, can also change the axis of rotation by 90° as desired. Note that the last element, the belt drive, may or may not be configured to change the axis of rotation and hence it will be considered as two different elements in the rest of this section.

STAGE 4: The updated requirements diagrams for each of the 12 newly-augmented partial designs are shown in Figure 3-11. Figure 3-11a represents the designs not including a bevel- or worm gear set or an angled belt drive (to make a right angle) and Figure 3-11b represents the ones that do include one of these elements.

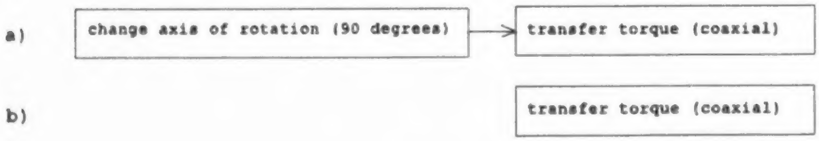

a)
change axis of rotation (90 degrees)
transfer torque (coaxial)
b)
transfer torque (coaxial)

Figure 3-11: Updated requirements diagrams of partial designs

STAGE 5: The updated search tree of the problem is shown in Figure 3-12. Note that for the "electric motor" alternative (left branch), the system has found two instances of one element (spur gear set) and for the "electric gear-motor" alternative (right branch), it has similarly found two instances of another element (bevel gear set). Also note that the system treats a dual-function element (the belt drive) as two separate elements (one that changes the axis of rotation and one that does not). As for the requirements diagrams, alternatives (1-3), (1-7), (1-8), (1-9), (2-12), (2-13), (2-13), (2-15), (2-16) and (2-17) in Figure 3-12 have the requirements FBD of Figure 3-11b associated with them whereas the rest of the alternatives have the diagram shown in Figure 3-11a for their requirements diagram.

STAGE 6: Exploring the knowledge pools of the 16 partial designs presented in Figure 3-12 reveals that those designs containing a belt drive or a chain drive violate the size-constraints of a hand-held drill (because of their relatively large centre distances). Hence nodes 8, 9, 10, 16, 17 and 18 are removed from the tree. The new search tree is shown in Figure 3-13.

STAGE 7: As the requirements diagrams of Figure 3-11 show, there are still unsatisfied elemental functions, so the system loops back to stage 2 again.

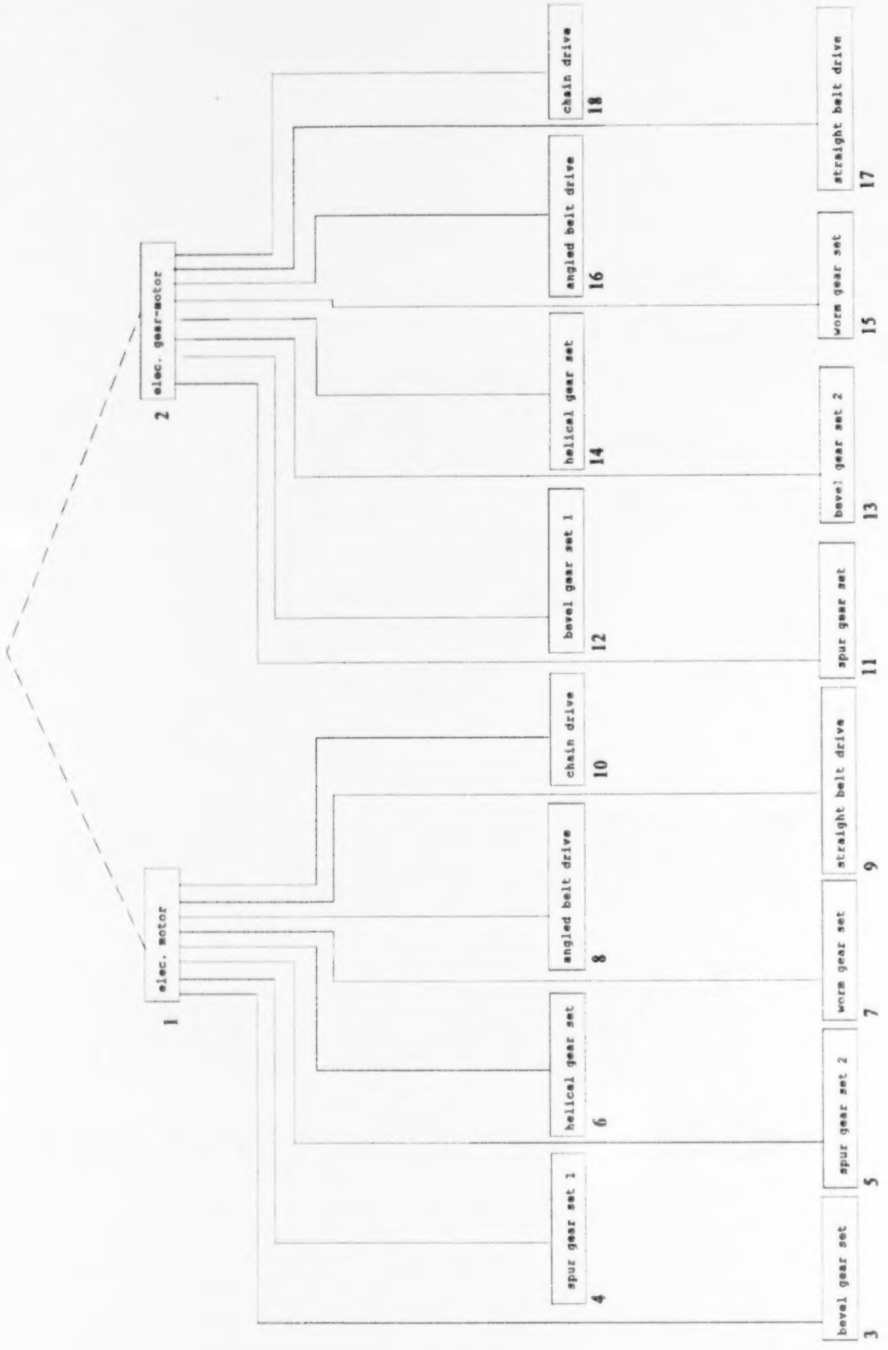

1 elec. motor
2 elec. gear-motor
3 bevel gear set
4 spur gear set 1
5 spur gear set 2
6 helical gear set
7 worm gear set
8 angled belt drive
9 straight belt drive
10 chain drive
11 spur gear set
12 bevel gear set 1
13 bevel gear set 2
14 helical gear set
15 worm gear set
16 angled belt drive
17 straight belt drive
18 chain drive

Figure 3-12: Partial search tree of the drill

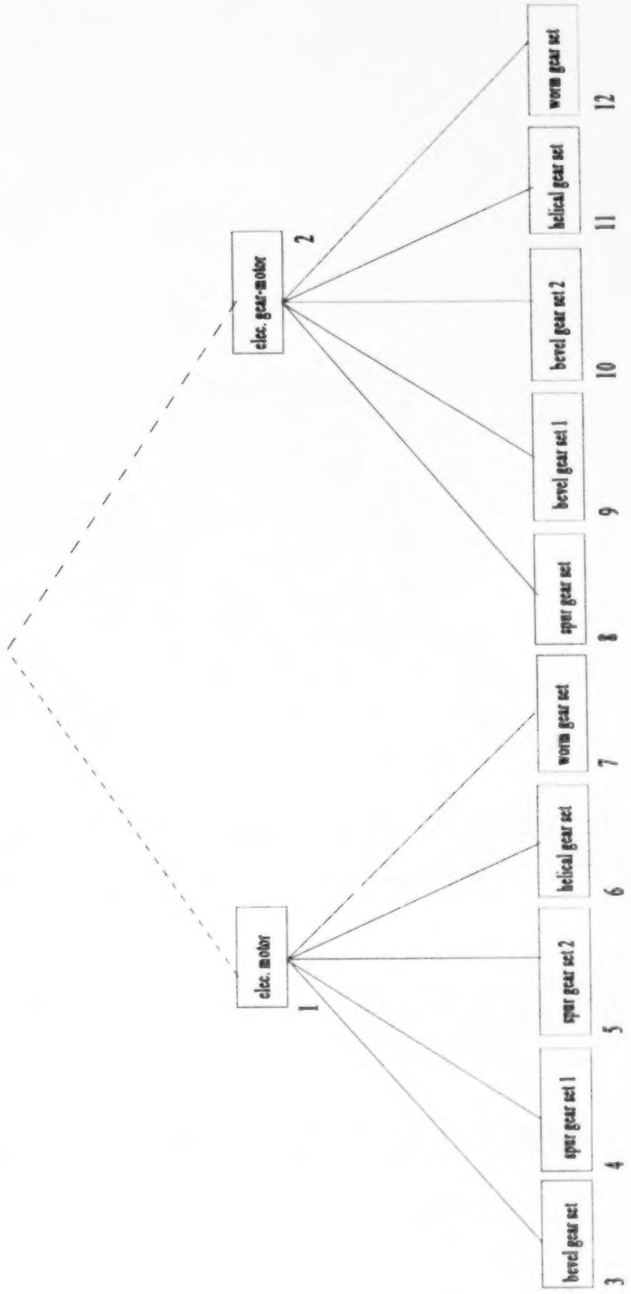

elec. motor
1
elec. gear-motor
2
bevel gear set
3
spur gear set 1
4
spur gear set 2
5
helical gear set
6
worm gear set
7
spur gear set
8
bevel gear set 1
9
bevel gear set 2
10
helical gear set
11
worm gear set
12

Figure 3-13: Pruned search tree of the drill

Let us, for brevity, skip the next two iterations and consider the final search tree of the drill (Figure 3-14) which shows 25 feasible design alternatives for this over-simplified problem. The results indicate that the system has found multiple instances of some elements (e.g. the worm-gear set, nodes 12 and 13, the bevel gear set, nodes 19 and 20 and the chuck, nodes 23 and 24). They also indicate that some alternatives have been rejected by the system's constraint-evaluation mechanism (e.g. the one comprising electric motor-spur gear set 2-worm gear set-chuck 1).

Each of the 25 final designs satisfies the initial functional requirements as well as the system's consistency constraints and any other constraints introduced during the course of design. These design alternatives can now be assessed for weight, cost, number of components or any other preferences of the user, and the optimal one(s) can thus be determined. For example, if we choose "number of components" as the optimality criteria, designs represented by paths (1-3-22), (1-6-32), (1-6-33), (2-9-38), (2-10-39), (2-12-45) and (2-12-46) will be of highest qualifications, whereas if "cost" is the major concern, designs (1-3-22), (1-6-32) and (1-6-33) will probably dominate the other alternatives (as an electric motor is usually less expensive than an electric gear-motor).

As we pointed out earlier, the design-by-exploration model prescribes the generation of *all* feasible solutions to the problem[4]. The task is practically impossible to achieve for a human designer in case of highly- or even moderately complex problems due to the size of the search space. We believe that the proposed approach to computer-based conceptual design will foster new design ideas by giving the user the chance to consider all his/her options, including the ones normally overlooked due to the human-designers' inevitable, experience-based biases.

---

[4] Note that here we are discussing the model at a theoretical level. In practice, however, it will normally take an enormous amount of computer time and memory to keep track of *all* solution alternatives where many of them are distinguished only by minor differences in details. This means that in the trade-off between the efficient utilization of computer resources and the universality of the model we will have to be selective in our generation and choice of design variants. In Chapter 5 we shall present techniques to avoid this type of combinatorial explosion and to keep the search-tree at a managable size.

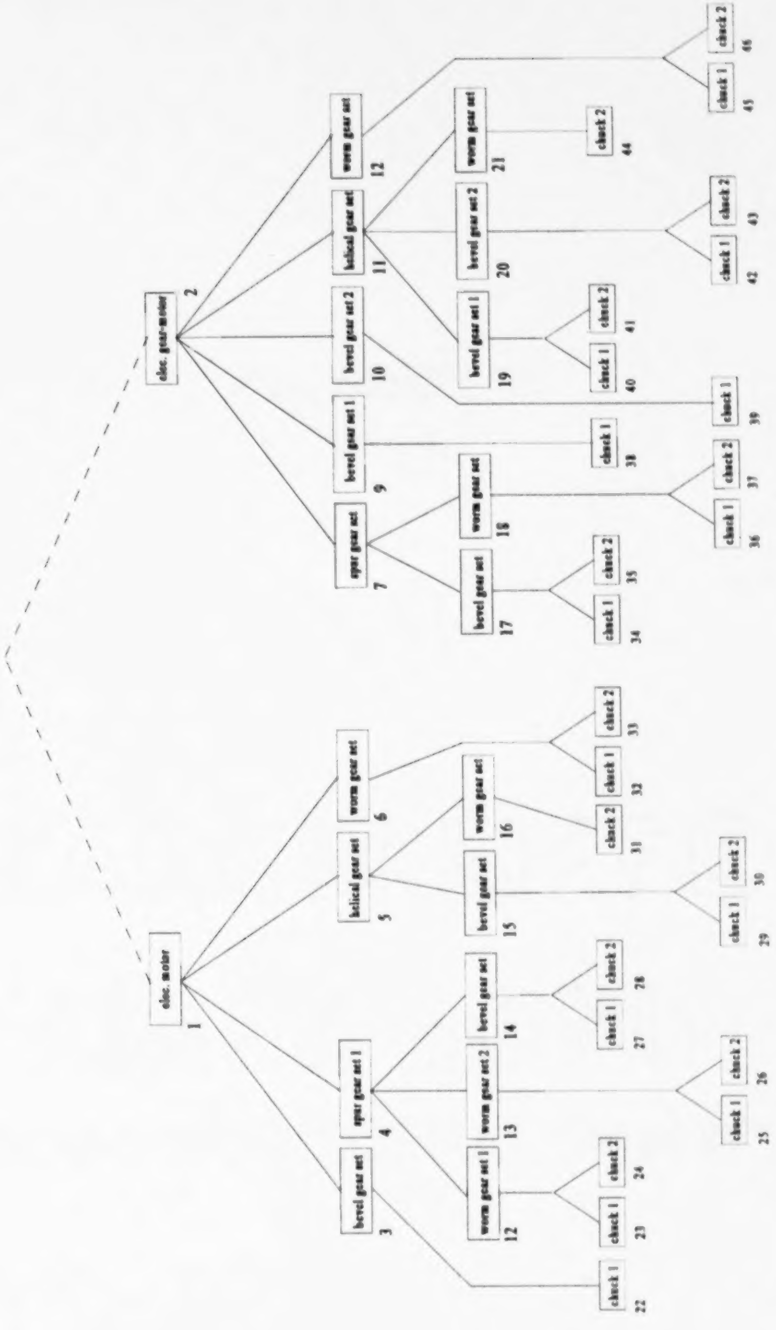

elec. motor
1
elec. gear-motor
2
bevel gear set
3
spur gear set 1
4
helical gear set
5
worm gear set
6
spur gear set
7
bevel gear set 1
9
bevel gear set 2
10
helical gear set
11
worm gear set
12
worm gear set 1
12
worm gear set 2
13
bevel gear set
14
bevel gear set
15
worm gear set
16
bevel gear set
17
worm gear set
18
bevel gear set 1
19
bevel gear set 2
20
worm gear set
21
chuck 1
22
chuck 1
23
chuck 2
24
chuck 1
25
chuck 2
26
chuck 1
27
chuck 2
28
chuck 1
29
chuck 2
30
chuck 2
31
chuck 1
32
chuck 2
33
chuck 1
34
chuck 2
35
chuck 1
36
chuck 2
37
chuck 1
38
chuck 1
39
chuck 1
40
chuck 2
41
chuck 1
42
chuck 2
43
chuck 2
44
chuck 1
45
chuck 2
46

Figure 3-14: Pruned search tree of the drill

3.3 CONVERGENCE AND COMPLETENESS CONSIDERATIONS

A question arises here as whether the iterative process presented in this chapter would get trapped in an infinite loop. The question stems from the assumption that the design system may never be able to completely "cover" all initial requirements, and never "get caught" by the constraint-evaluation mechanism either because it does not actually "violate" any constraints. This hypothetical situation would make the system loop back and forth in an attempt to find competent elements to carry out its uncovered functions.

We argue that this will never happen, because

- both the "requirements set " and the "mechanical elements set"[5] are finite sets,
- for each partial design, each uncovered elemental function will be considered only once,
- for each uncovered elemental function in the requirements set, the elements set is scanned only once; meaning that if all the elements in the knowledge-base of the system are considered once but none would carry out the desired function, the system will abandon the partial design under consideration and quit.

Based on this argument one of two things will happen *in a finite number of iterations:* either the system will come up with a number of feasible solutions to the problem (with or without relaxing some of the initial specifications), or it will quit and report that no such solution could be found under the given circumstances. If solution(s) are found *after* partially relaxing the initially given specifications, the system will report the new values of the adjusted variables as well.

Another, more essential question here is whether the Design by Exploration model guarantees to find *all* feasible solutions to the problem. The answer to this question, from a theoretical point of view, is yes. To elaborate this, consider the following definitions.

- $E = \{e_1, e_2, ..., e_n\}$ set of all existing physical elements,
- $S = \{s_1, s_2, ..., s_m\}$ set of all subsets of E (all possible combinations of e_i),
- $R = \{r_1, r_2, ..., r_p\}$ set of functional requirements of the problem,
- $C = \{c_1, c_2, ..., c_q\}$ set of the problem constraints,
- [illegible] $= \{d_1, d_2, ..., d_l\}$ set of designs (problem solutions) and
- $\underline{D} = \{\underline{d}_1, \underline{d}_2, ..., \underline{d}_k\}$ set of feasible designs.

Note that within the context of DbE model, a "design" is defined as a "collection of existing elements which meets certain requirements" and a "feasible design" is defined as a "collection of existing elements which meets certain requirements and satisfies certain constraints". Based on the definitions above, the following predicate-logic statements are true.

---

[5] One should keep in mind that throughout this work, by "mechanical elements set" we mean those elements contained in the system's knowledge-base.

a) $\forall\, d_i\ [d_i \subseteq E \wedge \text{satisfies}\,(d_i\,, R)] \rightarrow d_i \in D$

b) $\forall\, \underline{d}_i\ [\underline{d}_i \in D \wedge \text{satisfies}\,(\underline{d}_i\,, C)] \rightarrow \underline{d}_i \in \underline{D}$

c) $\underline{D} \subseteq D \subseteq S$

d) $\forall\, \underline{d}_i\ [\underline{d}_i \in \underline{D}] \rightarrow \underline{d}_i \subseteq E$

These statements express that each feasible design is a particular subset of the elements' set E, satisfying both R and C sets.

Various stages of the proposed model basically implement the statements above. We start out by considering S, the set of all possible element combinations, and apply R, the set of functional requirements, to spot and remove the subset of incompetent combinations of elements (stages 2 and 3). This will give us "D", the set of all (feasible/infeasible) designs. Then we apply the feasibility requirements "C" to this set to find the feasible subset of "D", represented by $\underline{D}$ (stage 6). So each feasible design can be considered a particular subset of the elements set E.

It should be clear by now that the selection mechanism of DbE will "sweep" all subsets of the elements' set E if there is no "screening" action involved. This is because at *each* level of the search tree the system considers *all* instances of *all* possible choices, which is, in effect, the same as an exhaustive search in covering the entire solution space. The screening action of the requirements/constraints is only to refute the incompetent/infeasible points *after* they have been considered, and not to make the system overlook part(s) of the solution space. Therefore the system will not, and cannot, miss a point in the solution space that matches our definition of a feasible design.

The discussion just presented examines the "completeness" question from a theoretical point of view only. There are, of necessity, practical limitations to the computer implementation of such a broad model. The next chapter is devoted to the elaboration of the problem and presentation of an implementation strategy to overcome these limitations.

3.4 SUMMARY

The *Design by Exploration* (DbE) model of mechanical conceptual design was described and its various stages were illustrated in the context of a simple design problem. The main characteristics of the model can be listed as follows.

- DbE conceptually designs those devices that can be described as "arrangements of existing physical elements".
- The model generates *multiple feasible,* rather than *single optimal,* design solutions.
- The model only generates the *functional spine* of the desired device, that is, a design generated according to DbE only satisfies the *given functional requirements* and not the

"supplementary requirements". This means that the model only transforms the given functional description of a device to a structural description and will not take the initiative to "improvise" other, sometimes obvious, supporting components (e.g. shaft and bearings for a gear drive).

- DbE is primarily intended as a paradigm for conceptual-design *automation* where user-dependency is kept at a minimum level. However, the model allows user/expert-modules interventions in the form of introducing new constraints (as we shall further discuss in the next chapter) at virtually any time during the design process.

CHAPTER 4

IMPLEMENTATION OF "DESIGN BY EXPLORATION"

The implementation of a model as broad as DbE involves the solution of many prerequisite problems. There are barriers to overcome and special-purpose means and methods to devise. The challenge we are facing mainly stems from two sources: the *dynamic nature of the design process* and the *automation requirements* embedded in the DbE model.

Design, in general, and conceptual design, in particular, are dynamic processes. *Information* as well as *Requirements* continue to flow into, and evolve within, the design system throughout the process. This is of course when one considers the real world problems rather than their over-simplified simulations, where both the "information" and the "requirements" of the problems remain unchanged throughout the process to result in reduced complexity.

To comply with the requirements of a dynamic design process, a design system must be able to:

- Collect, organize and interpret any given information as to what it means to the design,
- Accept any constraints introduced to it, regardless of where or when in the process it is being introduced,
- Adapt to the new situation, that is, make arrangements to verify its previous design decisions against the updated set of requirements and consider the set as the basis for its further decisions.

Moreover, the notion of *design automation* implies that the above tasks ought to be carried out, and all design decisions ought to be made, by the computer and with minimum or no need for user intervention.

With this introduction in mind, we start our discussion of the DbE's implementation issues by introducing our "problem-representation" plan. Meanwhile, we shall explain part of the terminology we shall be using in this chapter. The rest of the chapter will be devoted to outlining the architecture of an automated computer system (referred to as the *conceptual designer* henceforth) which embodies the Design by Exploration model. This will help us portray the environment in which various segments of our problem-solving strategy contribute to the design process.

We shall describe various components of the *conceptual designer,* the nature of their functions and the sequence of their actions to implement the steps prescribed by the DbE model. Our presentation, however, will not encompass an elaboration of the computational/reasoning techniques and procedures involved in the implementation. A detailed discussion of these will be presented in the next chapter.

4.1 PROBLEM REPRESENTATION

Before we can answer the question "how to solve a problem?" we need to answer another one: "how to formally represent a problem?"

It must be clear by now that the DbE model, and other mechanical conceptual-design models reported to date for that matter, are search-based. Despite their different solution strategies, they all start from an initial level of knowledge about a device (requirements and specifications) and seek a final level of knowledge (physical description of the device).

The tendency to use a search-based solution strategy in the context of mechanical conceptual design is quite natural. We recall the definition of a mechanical device as *a collection of physical components that collectively carries out a set of desired functions*. Hence each partial/complete design can be considered a special *subset* of the "physical components' set". Now suppose that each possible combination of (any number of) physical components could be represented by a spatial point (let us call the finite space thus generated the *solution space*). The solution space would then include the sought-after "special component-subsets" (or "designs" as we just defined). Therefore the design process can be best described as a search for feasible point(s) in the solution space, i.e. the ones that satisfy certain requirements[1].

The *solution space* just defined is sometimes called the *state space,* as each point in it represents not only a combination of physical elements but also a *state of knowledge* about the design. In the context of conceptual design, each "state of knowledge" tells us which requirements have been already satisfied, what is left to be done, and in short "how close to the desired state" we are.

The state-space representation seems natural for the conceptual design problem because the set of partial designs can be well organized in the form of a search tree. This is greatly to our advantage because it enables us to use some of the well-established AI techniques, as we shall further discuss in the next section.

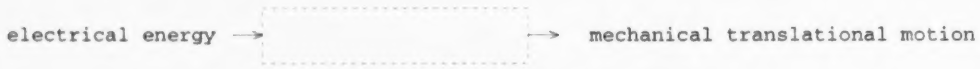


Figure 4-1: A simple problem representation

---

[1] The notion of *solution space* and the description of design as *a search in the solution space* will be further clarified in the rest of this chapter as well as in Chapter 5.

To further illuminate the definitions just presented, let us consider a simple example. Suppose that we have been asked to design a device to transform electrical energy (electricity) to mechanical energy (translational motion), say, in a forging press (Figure 4-1). The initial state of knowledge consists of a requirement (transform electric energy to mechanical translational motion) and the final state of knowledge will be the structural description of the device to carry out the task. For simplicity, let us alternatively state the problem as "to find a number of physical elements to convert the input quantity (electric energy) to the output quantity (translational motion)", that is, to reveal the contents of the "black box" in Figure 4-1.

The solution space, or state space, in this case consists of all partial and complete solutions to the problem. For instance, all physical components whose input quantity is electric power as well as those whose output quantity is translational motion represent points in the solution space.

4.2 SEARCH/REASONING STRATEGY

In the previous section we defined "problem solving" as seeking a path from an initial state to a goal state in the solution space of the problem. In the example of Figure 4-1, for

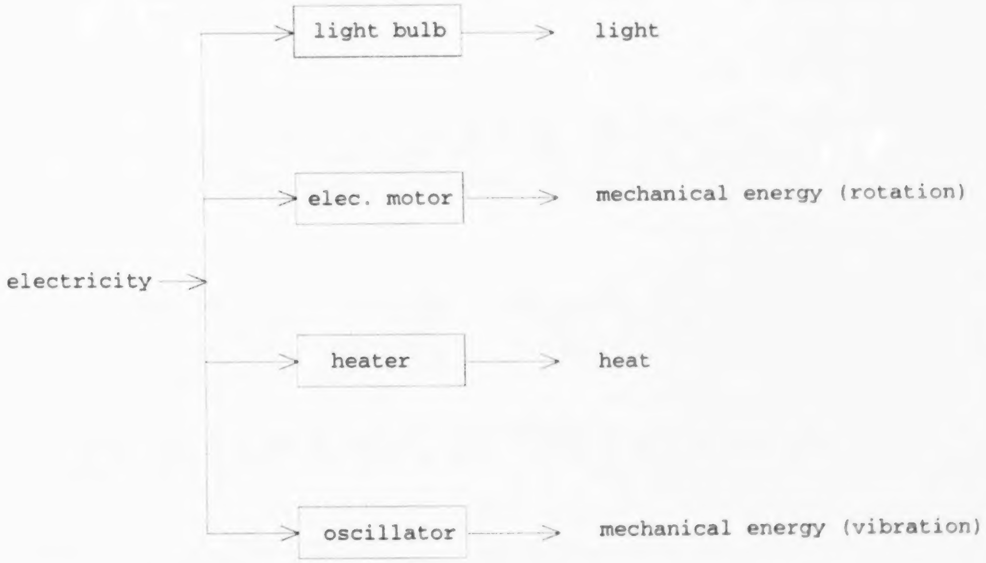
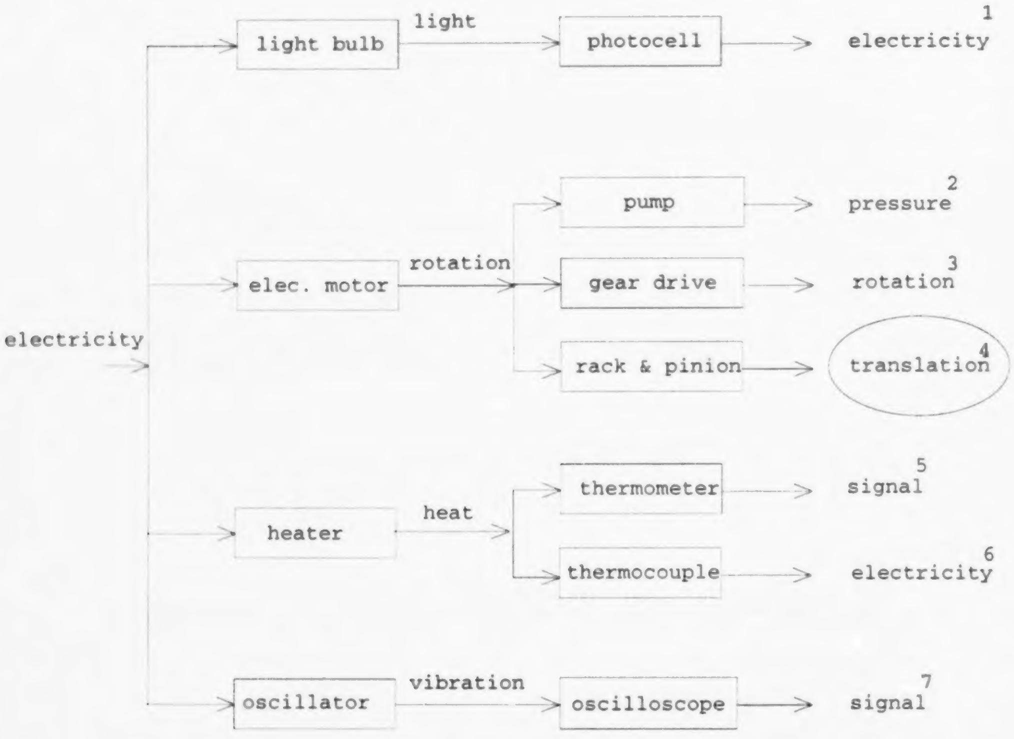


Figure 4-2: Forward search; first step

instance, a "path" can be visualized as an arrangement of components through which the input quantity is transformed to the output quantity. To find such a "path" one needs to build a search tree in the solution space in order to find the branch(es) connecting the initial state (node) to the final state. A *search-* or *reasoning model* specifies the way one organizes the reasoning steps and domain knowledge to search the solution space.

Several search models have been proposed, each of which suits a certain class of problems. In *forward search* one begins with the initial state at the root of the search tree and expands the tree until a (number of) leaf node(s) is (are) generated that match the desired final state. A node "qualifies" as a tree node if its input matches the output from one of the nodes in the previous level of the hierarchy. In a rule-based forward reasoning model (where if-then rules govern the search), this means that the next rule to be considered is one whose "if" part matches the "then" part of a previous rule.



electricity
light bulb
light
photocell
1
electricity
elec. motor
rotation
pump
2
pressure
gear drive
3
rotation
rack & pinion
4
translation
heater
heat
thermometer
5
signal
thermocouple
6
electricity
oscillator
vibration
oscilloscope
7
signal

Figure 4-3: Forward search; partially complete tree

In our example we start building the tree by finding all the elements with electricity as their input quantity (Figure 4-2). We then look for all the elements whose input quantities are the output quantities of the previous (first level) elements, and so forth. The process continues until the output(s) of some node(s) is (are) the desired quantity (e.g. output number 4 in Figure 4-3).

In *backward search,* one begins with the goal (desired) state at the root of the search tree and builds the tree towards the initial state. Here a node "qualifies" as a tree node if its output matches the input to one of the nodes in the previous level of the hierarchy. Similarly, in a rule-based backward reasoning model this means that the next rule to be considered is one whose "then" part matches the "if" part of a previous rule.

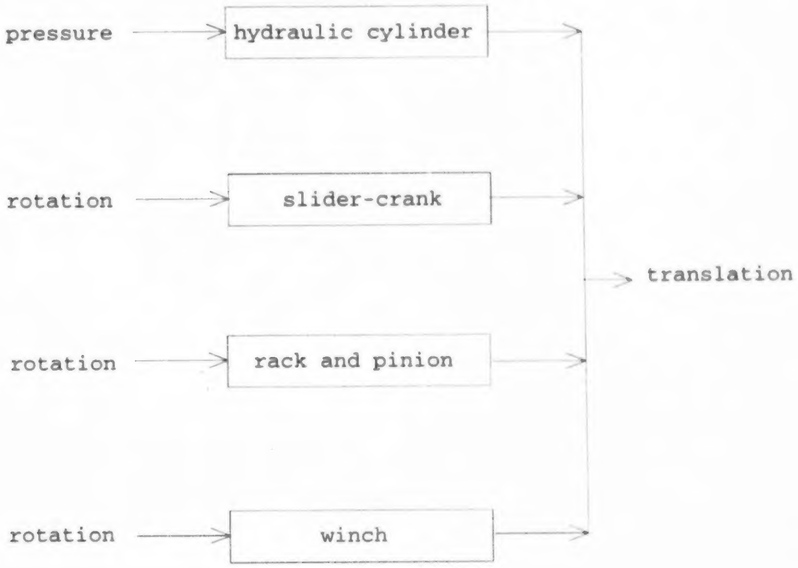


Figure 4-4: Backward search; first step

In the example of Figure 4-1, this time we start building the tree by finding all the elements with "translational motion" as their output quantity (Figure 4-4). We then look for all the elements whose output quantities are the input quantities of the previous (first level) elements, and so forth. The process continues until the input(s) of some node(s) is (are) the desired quantity "electricity" (e.g. inputs number 4 and 6 in Figure 4-5).

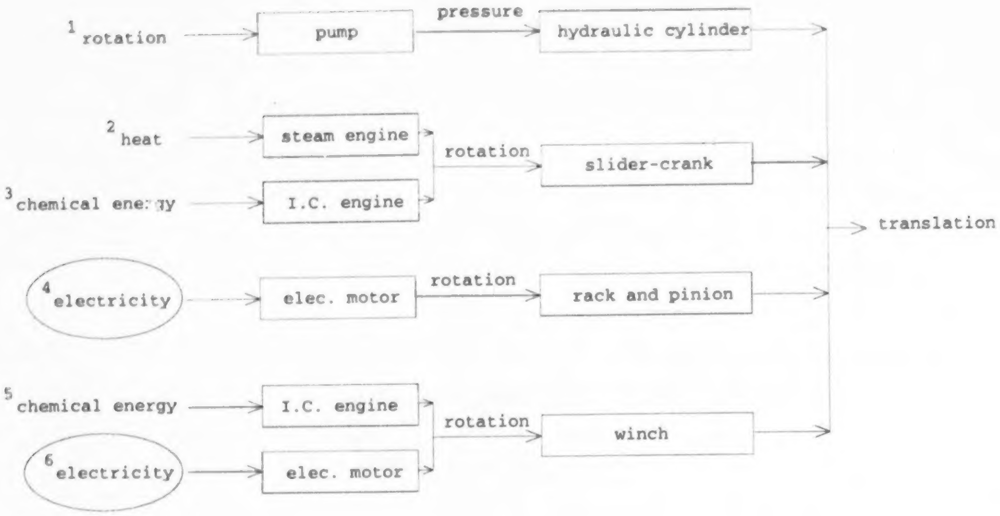

1 rotation
pump
pressure
hydraulic cylinder
2 heat
steam engine
rotation
slider-crank
3 chemical energy
I.C. engine
translation
4 electricity
elec. motor
rotation
rack and pinion
5 chemical energy
I.C. engine
rotation
winch
6 electricity
elec. motor

Figure 4-5: Backward search; partially complete tree

In *opportunistic search* a unique search direction is not maintained. One might move forward or backward at the most "opportune" time. It might be necessary at times to backtrack and rebuild/switch tree-branches in order to get "closer" to the goal state.

Briefly then, the central issue of problem-solving models is the question: what pieces of knowledge should be applied, when and how? In answer to this question a problem-solving model "provides a conceptual framework for organizing knowledge and a strategy for applying that knowledge." (Engelmore, Morgan and Nii 1988)

None of the models just introduced qualifies as a sole problem-solving strategy for the DbE conceptual-design model. Our discussion of the DbE model in the previous chapter implies that:

- because of the complex structure of the solution space and also because of the continual change of problem knowledge (specifications and constraints), a monotonic search is likely to fail to find a feasible path to the goal state;
- the problem-solving model must be able to handle multiple sources of knowledge, including the user, the *design engine*[2] and the expert modules.

To explain what we mean by the latter, let us visualize the design space as a multidimensional space in which each dimension represents a different lifecycle objective such as function, fabrication, serviceability, reliability and cost. A feasible point in this space, i.e. a design solution, must simultaneously meet the requirements of all these objectives.

---

[2] The concept of a *Design Engine* will be elaborated in the next chapter.

The notion of *Concurrent Design* has been developed to help achieve this goal. It prescribes the concurrent, rather than sequential, consideration of various requirements of multiple lifecycle objectives. The task, although somewhat difficult to accomplish, is the most logical and efficient way to handle different, sometimes even conflicting, requirements. It shortens the design process by unveiling the inconsistencies and hence rejecting the infeasible alternatives at an early stage. It also helps avoid the loss-of-the-overall-optimal-design syndrome which commonly occurs in systems based on a best-first selection strategy as explained in chapter 2. In concurrent design this is avoided by endorsing at each stage only those alternatives that satisfy requirements from all applicable sources simultaneously, regardless of whether or not they are local optima.

Although in this work we have been and we will be primarily concerned with *function* and *form* of mechanical systems among other design considerations, the problem-solving model we are about to introduce will nevertheless support *concurrent design*. It will lay the foundations of an automated system flexible enough to host various objectives of a device's lifecycle.

In the previous chapter we mentioned that the DbE model allows for multiple sources of knowledge to contribute to the design process. We referred to some of these sources as *expert modules*. Expert modules represent various lifecycle objectives in design. As we shall see later on, each E. M. (e.g. the manufacture E. M., the reliability E. M. or the cost E. M.) will be able to "watch" a design as it evolves through the design process and may "criticize" the design by verifying it against the requirements of the objective it represents. For example, the "cost" expert module will be able to check the total cost of each partial design alternative against its feasibility criterion and reject the infeasible ones virtually anytime during the design process. The most common way for the expert modules to contribute to a design is through a) verification of the partial designs and b) imposing new constraints on the problem.

The above discussion was meant to answer the question "why do we consider none of the three search strategies outlined at the beginning of this section adequate for the implementation of DbE model?" As we just explained, the model relies on the simultaneous exploitation of multiple knowledge sources in an improvised order. Also from our discussion of the DbE model in chapter 3 it can be inferred that the model advocates a special form of an opportunistic search. These conditions are best suited to the *blackboard model* of problem solving.

4-3 THE *BLACKBOARD* MODEL OF PROBLEM SOLVING

As we just explained, the problem of mechanical conceptual design, in its general form, is a complex, multidisciplinary problem requiring the cooperation of experts in various aspects of the product's lifecycle. The blackboard model of problem solving provides just the right environment for that.

The idea behind the model is that of a group of experts who use a blackboard as a communication medium during a brain-storm. Being a special case of opportunistic problem solving, the model allows various expert modules to contribute to the evolution of the design whenever their expertise is required. We shall talk about the experts involved in the development of the conceptual designer shortly.

To better portray the notion of a blackboard system, let us draw an analogy between the operation of one such system and that of a group of human problem-solvers. Imagine a group of people trying to solve a complicated jigsaw puzzle made out of a city map. The puzzle may have several pieces of the same size and shape, so other characteristics of the pieces than "shape" and "size" must be considered as well.

The group starts out by arbitrarily putting a piece on a puzzle board which can be seen/accessed by all participants at all times. They then start suggesting pieces that seem to fit in. In the meantime the whole group, supposedly composed of experts, is watching the evolution of the solution. The "shape/size" expert continually checks the partial solution and spots those pieces that do not fit in shape-wise and/or size-wise. Even if a piece does fit in physically, there are still chances that it has been misplaced as it could represent the wrong neighborhood. When this happens, the "neighborhood" expert steps forward and voids the incorrect move and removes the piece from the board. Going yet farther, the "street" expert continually keeps an eye on the partially complete map to make sure the pieces are placed correctly. The process continues until the puzzle is complete, which means all the experts certify its completeness and correctness.

Having considered the simplified analogy above, we can now point out the main characteristics of the blackboard model of problem solving. But first we need to define some terms we shall be using in our discussion.

- *Problem knowledge* is a general term representing all the information we are given about the problem. It includes given specifications and domain-specific knowledge as well as all the requirements and constraints of the problem. In general, the problem knowledge can be qualitative or quantitative, numerical or non-numerical. Dimensions of a shaft, required output torque of a gearbox, design equations of a belt drive, recommended material for a tension spring and reliability requirements of a machine tool can all be pieces of "problem knowledge".

- By a *knowledge source* we generally mean the person/software-module that provides/processes part of the problem knowledge. It could be a computer user, a domain expert, a database, a structured collection of if-then rules or a software module containing design equations of a component, among other things. Nonetheless, in the context of the blackboard model the term knowledge-source (KS) refers to a subset of the above, namely an *expert software-module,* as we shall see shortly.
- The term *panel* (in this case a blackboard panel) figuratively refers to a specific portion of a software unit (database, program, etc.). The unit could be partitioned into multiple panels that communicate/interact within the main unit.

Using the above terminology, we shall now outline the main characteristics of the blackboard model (Terry 1988; Kitzmiller and Jagannathan 1989; Erman et al. 1980; Nii 1986).

1) The model is a highly structured special case of opportunistic problem-solving. The analogy presented earlier clearly illustrates what we mean by "opportunistic". In that example, there was no *a priori* plan as to which piece would be posted on the board next and which expert would react to this move.
2) The model consists of two basic components (Figure 4-6): the *knowledge sources* and the *blackboard.* The problem knowledge is partitioned into a number of separate, independent software modules called "knowledge sources" or "expert modules", e.g. functions module, geometry module and cost module. Each of these modules is appointed to carry out a particular task and will use its expertise to contribute to the problem-solving process when required. A knowledge source can comprise a set of if-then rules or a computation procedure, among other things.

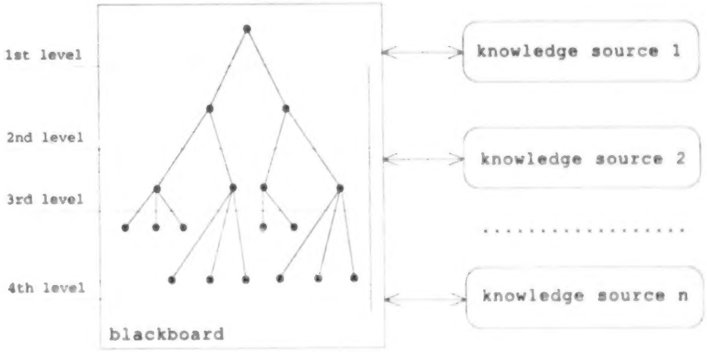


Figure 4-6: The *blackboard* model of problem-solving

3) In addition to the knowledge sources, there is a single global database, called blackboard, which continually provides an up-to-the-moment representation of the design and the latest state of the problem solving. Its purpose is to hold computational and solution-state information needed by and produced by the knowledge sources.

4) The blackboard is shared by and accessible to all the knowledge sources. They make changes to the contents of the blackboard and these changes lead incrementally to a (number of) solution(s). Communication and interaction among the knowledge sources take place solely through the blackboard.

5) The contents of the blackboard mainly consist of objects from the problem's solution space (described in section 4-1). These include partial/final designs and the information associated with them. The objects are hierarchically organized into levels of analysis. This organization perfectly matches that of DbE's search tree (section 4-2). Information about the objects on one level serves as input to one or more expert modules. These modules will process the information and generate new information to be placed on the same or the following levels. For example, in a typical implementation of DbE, information about one partial design in the search tree could be used by some "evaluation expert" to verify the validity of the design. The result ("accepted" or "rejected") would then be posted on the same level of the tree.

6) The blackboard can have multiple *panels,* that is, the solution space can be partitioned into multiple hierarchies. For example, in the context of mechanical design partial designs may be hierarchically organized according to their geometry (geometry panel) as well as their function (function panel). Also it is possible that some of the knowledge sources have access only to certain panels. For instance, a "tolerance" knowledge source may only access the geometry panel as it had nothing to do with a device's function.

7) The decision as to when each knowledge source should be activated and applied to the contents of the blackboard, is made dynamically. There is no prescribed agenda for the participation of the knowledge sources. There is, however, a control module that

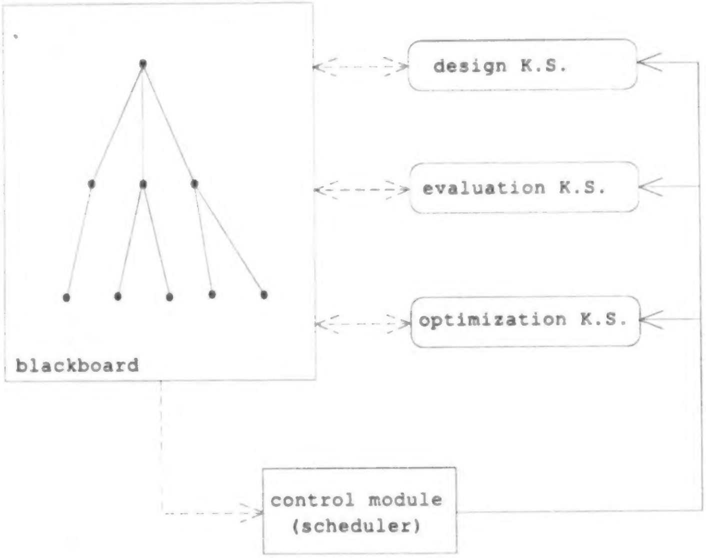


Figure 4-7: A simple blackboard system for the example

monitors the changes on the blackboard and decides whether the conditions for the activation of a certain knowledge source have been fulfilled. For example, the "cost" knowledge-source (expert) will not evaluate the costs of various design alternatives appearing on the blackboard unless the control module confirms that the individual costs of their components have been calculated/looked up.

Before we go ahead and describe the blackboard-based architecture of our *conceptual designer,* let us briefly re-examine the above characteristics in the context of our simple example of Figure 4-1. For this purpose we consider a simple blackboard system with a single-panel blackboard, a control module and three knowledge sources (expert modules): the *design expert,* the *evaluation expert* and the *optimization expert* (Figure 4-7).

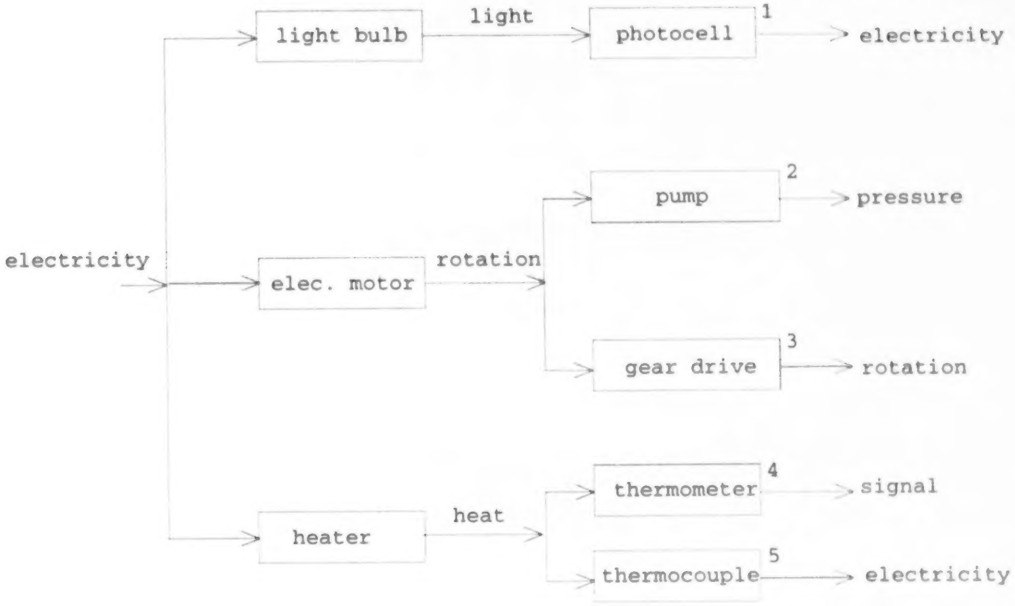


Figure 4-8: Hypothetical solution state for the example

Figure 4-8 shows the contents of the blackboard at a hypothetical solution state. At this point the control module (sometimes called the *scheduler*) decides that the design expert should be activated to generate the next level of the search tree, as the final goal (transform electricity to translation) has not been achieved yet. As a result of this activation, new partial designs 1, 2 and 3 are produced (Figure 4-9), through matching the output of node 1 (Figure 4-8) to the inputs of all elements contained in the system's library (forward search). Again the control module browses through the information on the updated blackboard and decides that more tree-nodes need to be generated because the current level of the search tree is not

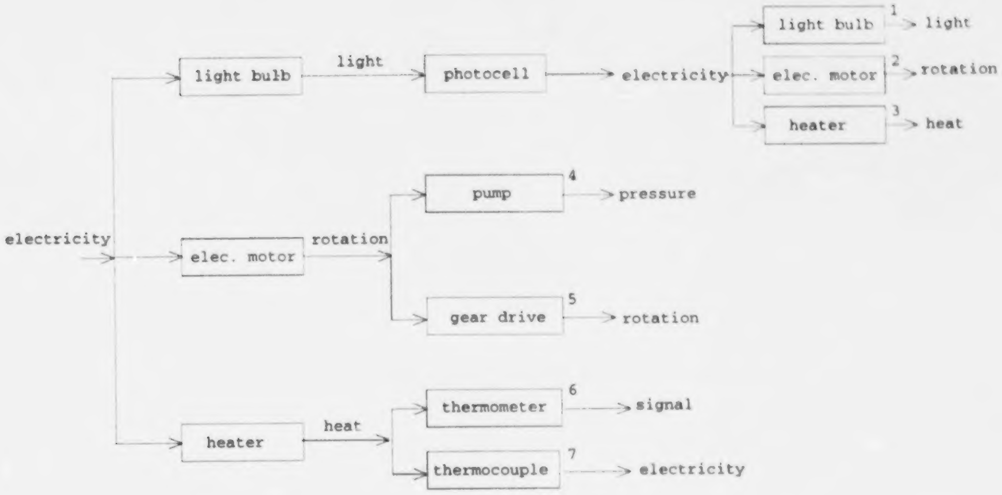

electricity
light bulb
light
photocell
electricity
light bulb
1
light
elec. motor
2
rotation
heater
3
heat
elec. motor
rotation
pump
4
pressure
gear drive
5
rotation
heater
heat
thermometer
6
signal
thermocouple
7
electricity

Figure 4-9: The solution state after first activation of the design expert

complete yet. The design knowledge-source is activated again and partial designs 4 to 10 are generated (Figure 4-10). Note that the number of times the design expert has to be activated could not be determined ahead of time. In this example it depended on the number of nodes in the previous level of the hierarchy as well as the number of matchings found for each node. This is an example of dynamic decision-making discussed earlier (item 7 above).

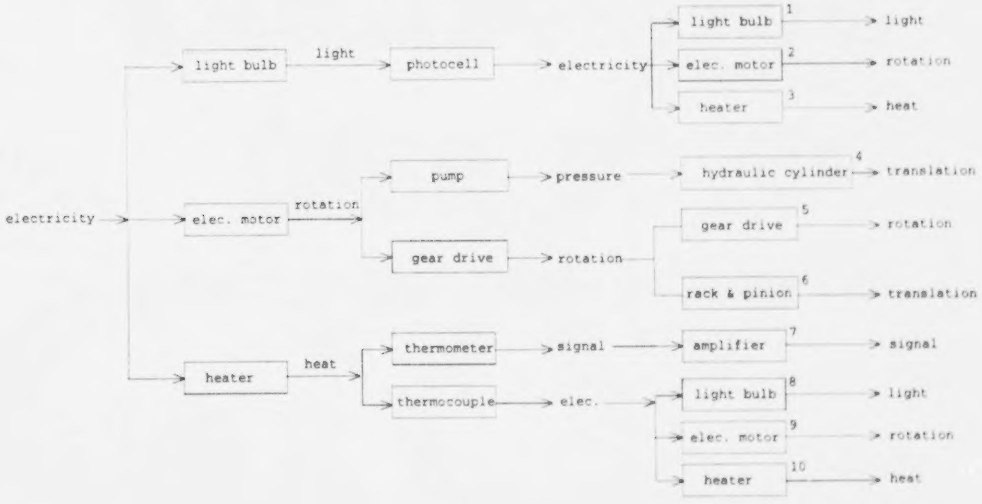

electricity
light bulb
light
photocell
electricity
light bulb
1
light
elec. motor
2
rotation
heater
3
heat
elec. motor
rotation
pump
pressure
hydraulic cylinder
4
translation
gear drive
rotation
gear drive
5
rotation
rack & pinion
6
translation
heater
heat
thermometer
signal
amplifier
7
signal
thermocouple
elec.
light bulb
8
light
elec. motor
9
rotation
heater
10
heat

Figure 4-10: The solution state after completion of the fourth level

Now that the outputs of two of the leaf-nodes (nodes 4 and 6 in Figure 4-10) match the desired goal (translation), the control module decides that the evaluation expert should be activated to verify the two designs against, say, consistency and efficiency criteria. The evaluation expert then accesses the global database (blackboard) and examines the partial designs one at a time.

Let us suppose, for the sake of brevity, that both suggested alternatives (namely the *electric motor-pump-hydraulic cylinder* and the *electric motor-gear drive-rack and pinion* configurations) satisfy the consistency requirements. This means that the pump can operate at the speed of the motor and that the pressure generated by the pump is within the acceptable range for the hydraulic cylinder. It also means that both the gear drive and the rack and pinion can operate at the speed of the electric motor.

The efficiency of the system represented by the path leading to node 4 (Figure 4-10) can be calculated by multiplying the efficiencies of the electric motor, the pump and the hydraulic cylinder. The resulting efficiency will be expectedly low due to the low efficiency of the pump. Comparing this efficiency to that of the other design (electric motor, gear drive and rack and pinion), the evaluation expert will choose the latter (the path leading to node 6 in Figure 4-10). Again note that the number of times the evaluation knowledge-source would be activated was not determined *a priori*.

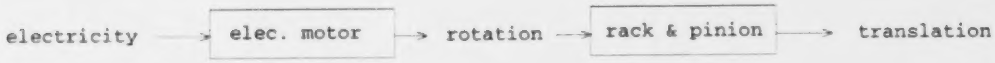


Figure 4-11: The optimal design generated by the optimization expert

At this stage the control module activates the optimization expert, as the conditions for that (finding one or more feasible designs) have been fulfilled. The optimization knowledge-source examines the respective path (the one leading to node 6 in Figure 4-10) in a backward search and finds two nodes with the same input (rotation) along it, namely the gear-drive node and the rack-and-pinion node. It then removes the redundant node (the gear-drive) and thus increases the efficiency while reducing the cost. Note that the optimization KS would not do this if the rack and pinion could not operate at the speed of the motor and the gear drive was needed to reduce the speed first. The final outcome is then presented as the optimal design (Figure 4-11) and the process is terminated.

The above example, though not a real design problem, illustrated some of the basic characteristics of the blackboard model. High level of modularity (i.e. the problem knowledge being organized in separate, independent modules), opportunistic search strategy, dynamic decision making and concurrency are among these characteristics.

4.4 *CONCEPTUAL DESIGNER:* THE SYSTEM ARCHITECTURE

Our discussion of the blackboard model in the last section was aimed at justifying our choice of the model for the implementation of the Design by Exploration in the form of a computer system called the *Conceptual Designer*. It will be further shown in this section that the main requirements of this implementation will be well satisfied by the choice of a blackboard-based system architecture.

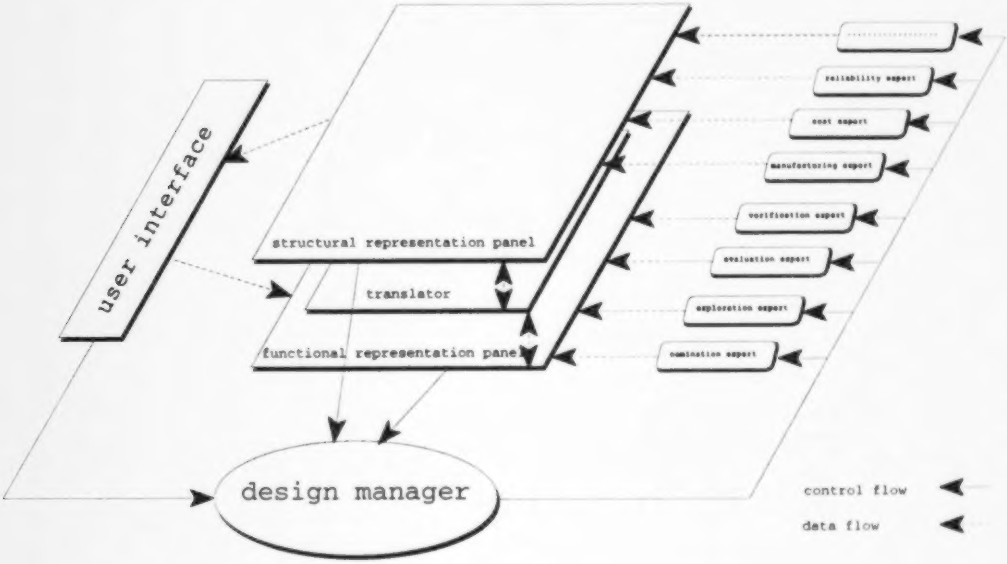

user interface
structural representation panel
translator
functional representation panel
reliability expert
cost expert
manufacturing expert
verification expert
evaluation expert
exploration expert
nomination expert
design manager
control flow
data flow

Figure 4-12: System architecture of the *Conceptual Designer*

Illustrated in Figure 4-12 is the framework of the *conceptual designer*. In addition to the two main components "blackboard" and "knowledge sources", the system includes a user interface which enables the user to communicate with the system. It basically plays an interpreter that translates the information/requirements provided by the user to the language of the system, that is, to a format understandable by the knowledge-processing units. The interface also presents the final results and/or any other messages of the system in a format which can be appreciated by the user.

The control module (scheduler) of the system is called the *design manager.* Its main job, as explained earlier, is to coordinate the activation of various knowledge sources. The control flow between the design manager and the knowledge sources, as well as the data flow among various components of the system is shown in Figure 4-12.

The system has a two-panel blackboard, meaning that the solution space is partitioned into two hierarchies. As we shall explain shortly, the two hierarchies basically represent two "views" of the same search tree. Due to the nature of their contents, the two blackboard panels are acted on by two different sets of knowledge sources.

In the rest of this chapter we shall first elaborate the components of the system and their functions in the context of the DbE model and then emulate a typical problem-solving cycle to demonstrate the functioning of the system.

4.4.1 THE BLACKBOARD

Each of the two panels of the system's blackboard contains a different representation of the problem's search tree. We remember from our earlier discussions that the nodes of the search tree are partial/complete design alternatives. On the *structural representation panel* each node of the search tree presents a *structural description* of a partial design. The structural description (representation) of a design provides information about the object's constituent *elements* as well as the *configuration* and *interconnections* of these elements.

According to our *structural representation method* (Appendix B), the formal physical description of a device is composed of two parts: *graphical representation* and *symbolic representation.* The former is used to communicate the generated designs with the user whereas the latter is primarily meant for the internal use of the system. The two types of representations are illustrated in Figure 4-13 which shows the simplified contents of a typical *structural representation panel.* To avoid unnecessary complexity at this demonstration stage, in the figure we have replaced the formal graphical representations of the objects with simple component-icons accompanied by the related numerical information.

Shown in Figure 4-13 is the *structural representation panel* pertaining to a hypothetical stage in the solution of the simple-drill design example of Section 3.2. We recall from there that at some stage, the computer system found three candidate components to partially satisfy the requirements of the problem, i.e. rotating a drill-bit at a certain speed and torque. These components were an "electric motor", an "electric gear-motor" and an "internal combustion engine". Simplified physical descriptions of these components are presented in the figure.

Most of the information associated with the structural representation of a design is the kind of information needed for evaluating the *integrity* and *consistency* of that design.

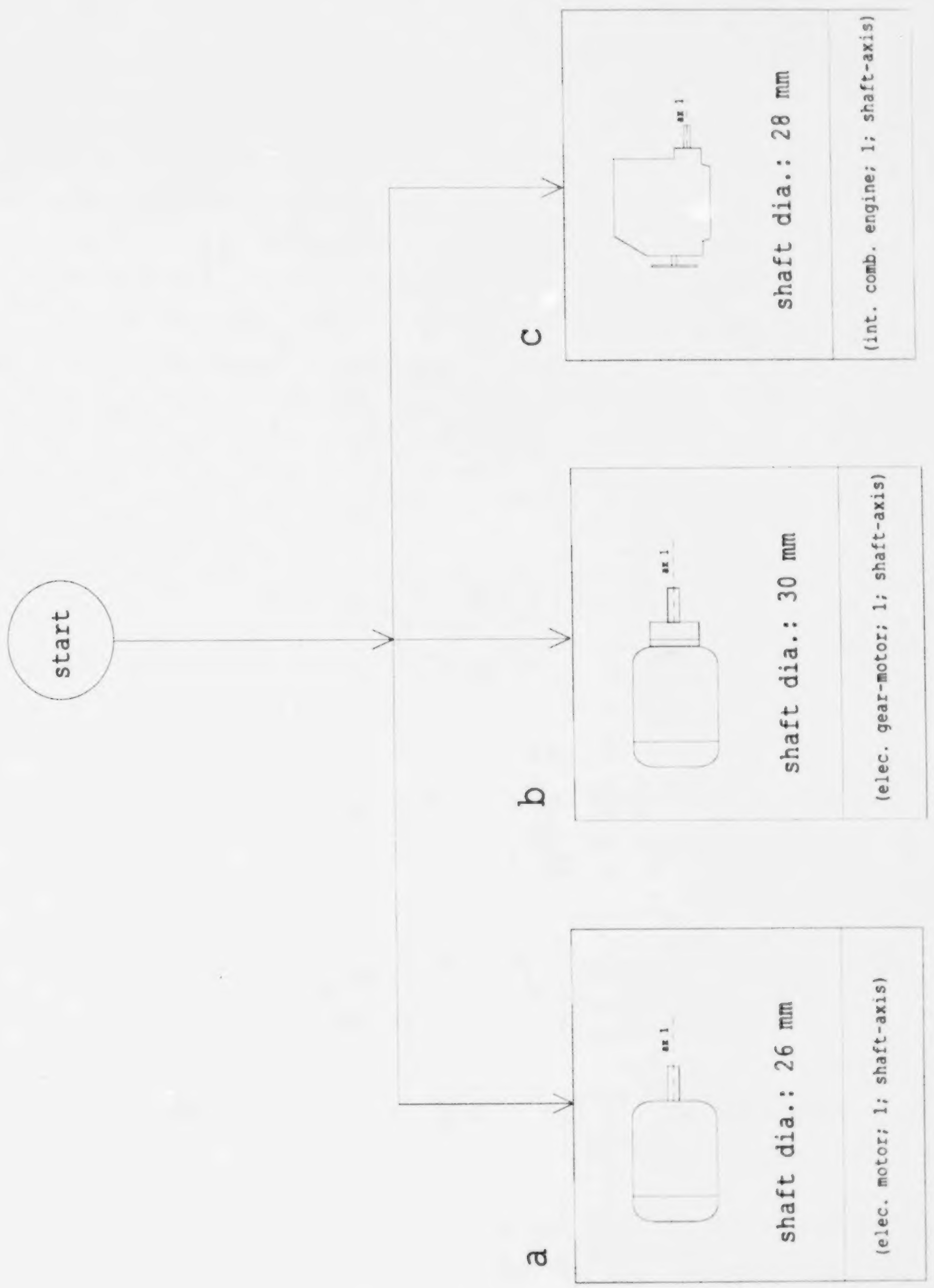

start
a
shaft dia.: 26 mm
(elec. motor; 1; shaft-axis)
ax 1
b
shaft dia.: 30 mm
(elec. gear-motor; 1; shaft-axis)
ax 1
c
shaft dia.: 28 mm
(int. comb. engine; 1; shaft-axis)
ax 1

Figure 4-13: Typical contents of the Structural-Representation Panel

This basically consists of the specifications of the "ports" of the design (Appendix B), whereby a component/subsystem connects to others. In the example above, shaft-diameters are specified for each of the three alternatives, as this information is essential to the choice of other components to be mounted on these shafts. In short, the ability of the design system to check the consistency of a design rests upon the information contained in its structural representation.

In addition to the information just pointed out, there are other types of information that can be included in the structural representation panel "on request". The nature of these other types of information highly depends on the expert-modules using them. For example if the system includes a "cost" expert-module, then the cost of each component must be included in the structural representation of the design. Or if a "serviceability" knowledge-source is involved, then the maintenance characteristics of the design must be provided as part of its structural representation.

The second panel of the blackboard, the *functional representation panel*, basically contains the same search tree as the structural representation panel, except that this time each node presents a *functional description* of the respective design. The functional description of a design is composed of two parts:

- the Function Block Diagram (Chapter 1) of the design, including the values of its design/performance parameters, and
- a second FBD representing the unsatisfied portion of the problem's requirements, plus the applicable information (specifications/constraints).

Figure 4-14 illustrates the contents of the blackboard's functional representation panel for the drill example introduced earlier. The panel has been shown at the same solution stage as the one illustrated in Figure 4-13. For brevity, the symbolic representation of the functions has been partially omitted.

The functional representation panel of the blackboard does not normally contain information regarding the *structure* of the designs it represents. Similarly, the structural representation panel normally contains no information regarding the *function* of the designs it represents. Ignoring some occasional, minor overlaps, one could say that the two panels are mutually exclusive with respect to the information they provide.

Other than containing different types of information, the two blackboard panels are also different in that they are accessible by two distinct sets of knowledge sources, as illustrated in Figure 4-12. Further explanation of this requires a brief introduction.

Looking at the performance of the *conceptual designer* from a "product" point of view (i.e. focusing on the *designed artifact* itself rather than the design process), one can divide the activities of the system into two distinct, yet interactive groups: *generating*

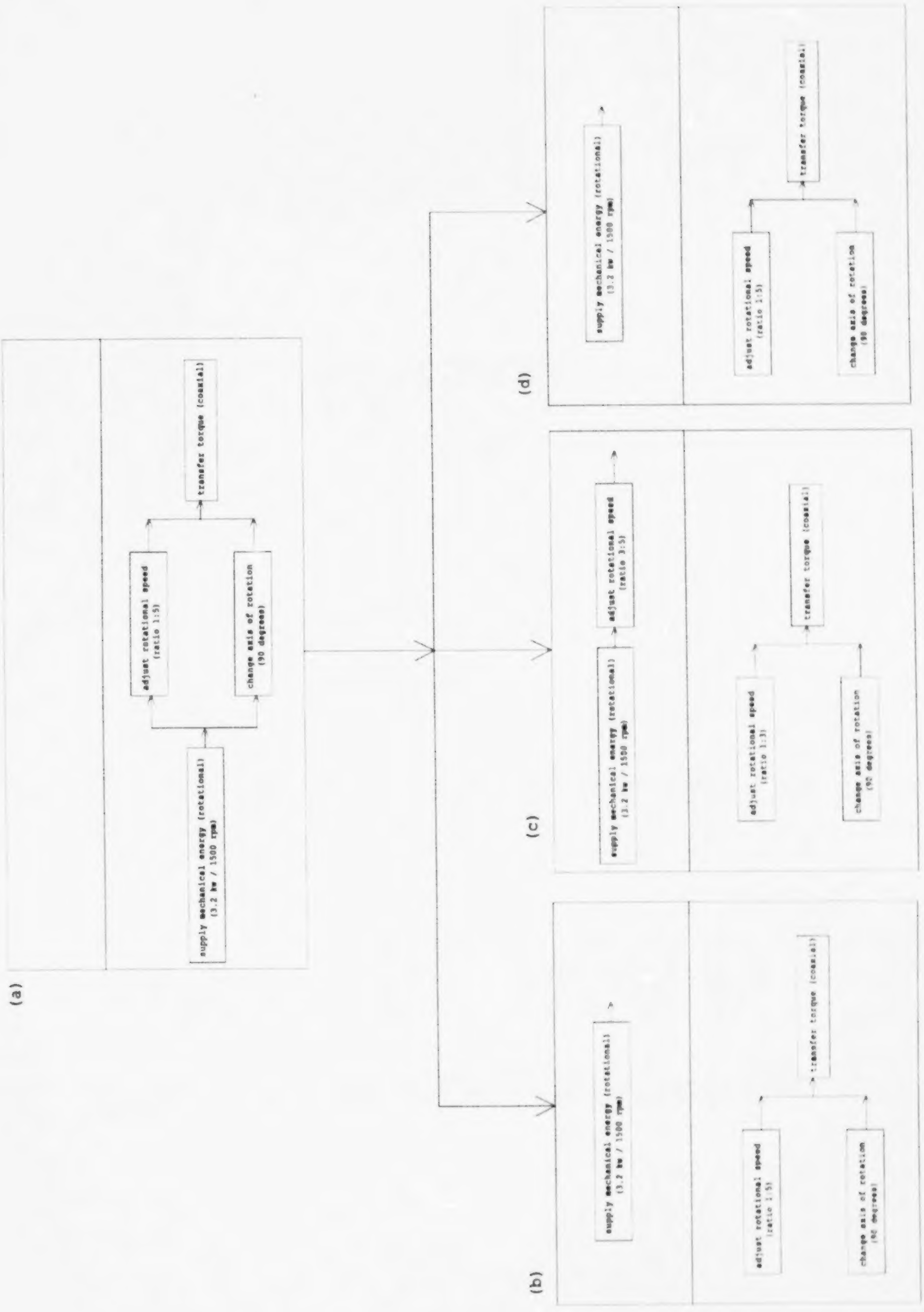

(a)
supply mechanical energy (rotational)
(3.2 kw / 1500 rpm)
adjust rotational speed
(ratio 1:5)
change axis of rotation
(90 degrees)
transfer torque (coaxial)
(b)
supply mechanical energy (rotational)
(3.2 kw / 1500 rpm)
adjust rotational speed
(ratio 1:5)
change axis of rotation
(90 degrees)
transfer torque (coaxial)
(c)
supply mechanical energy (rotational)
(3.2 kw / 1500 rpm)
adjust rotational speed
(ratio 3:5)
adjust rotational speed
(ratio 1:3)
change axis of rotation
(90 degrees)
transfer torque (coaxial)
(d)
supply mechanical energy (rotational)
(3.2 kw / 1500 rpm)
adjust rotational speed
(ratio 1:5)
change axis of rotation
(90 degrees)
transfer torque (coaxial)

Figure 4-14: Typical contents of the Functional-Representation Panel

activities and *criticizing* activities.

We remember from our discussions in chapter 1 that the DbE model initially prescribes the generation of a *candidate* design based on the knowledge then available. This is what we mean by a "design activity". The model then calls for the verification of this candidate design based on the emerging information about the problem. This is what we mean by "criticizing activity". The verified design would be presented as the feasible solution to the problem.

"Criticism" is used here as a more general term for "verification". This is because according to the notion of concurrent design, each expert module not only checks the *feasibility* of a design but may suggest modifications to improve the quality of the design according to its own standards.

With this introduction behind us, we can now get back to the system's blackboard and discuss the rationale behind separating its two panels. Generally speaking, the functional representation panel is the sub-blackboard for the *generating* activities, whereas the structural representation panel is the sub-blackboard for the *criticizing* activities. In other words, the functional representation panel contains the information produced by, and used by the knowledge-sources involved in generating the initial candidate designs. This becomes more clear when we remember that "functions" are the dominant players in early stages of our scenario for conceptual design, namely "component selection" and "exploration".

By the same token the structural representation panel contains the information produced by, and used by the knowledge-sources involved in evaluating and criticizing candidate designs. This assertion is based on the observation that most of the "critic" modules require structural, rather than functional, specifications of the designs, e.g. dimensions, weight, cost and maintenance requirements. Nevertheless, the *conceptual designer*'s architecture has the flexibility to let the critic modules access the functional representation panel on request.

We shall say more about the two panels in the following subsection where we discuss the knowledge-sources.

In addition to the two panels discussed above, the blackboard includes a "translator" layer which maps the contents of one panel onto the other. As prescribed by the DbE model, design decisions involved in the *generating* activities will be initially posted on the functional representation panel. The *translator* layer then immediately "extracts" the structural specifications of the posted design and posts them on the structural representation panel. Similarly, any decisions (e.g. rejection of a partial design by the cost-expert) or new requirements (e.g. a constraint limiting the total weight of the design by the weight-expert) posted on the structural representation panel will be immediately reflected on the functional representation panel. This is mainly done by removing the rejected design from the search tree or by adding the new constraints to the constraint sets of all partial designs on the functional representation panel.

The use of a "translator" layer will guarantee the synchronous update of the two blackboard panels and that all the knowledge-sources will be accessing the same solution state at all times.

4.4.2 THE KNOWLEDGE SOURCES (KS's)

A knowledge source (expert module) is a specialized software module which uses its domain "expertise" to modify the information contents of the blackboard. In the context of the *conceptual designer,* this modification is intended to improve the qualifications of the partial designs incrementally, and thus bring the system closer to finding the final solution(s).

A knowledge source, as far as this work is concerned, is either a *procedure* or a *set of rules* or a combination of both. A *procedure* is a sequence of computational/reasoning steps taking place in a preset order. Figure 4-15 shows the flow-diagram of a procedure to find the critical points of a function of the form Y = F(x).

A *rule-base* on the other hand, is a set of *if-then* rules which, upon activation, seeks to match the situation on hand with the condition (if) part of some rule(s) it contains. If successful, it then posts the result (then) part of the rule(s) on the blackboard. Figure 4-16 presents part of a rule-base (in predicate logic) which acts as a "size" expert module. Its job is to make sure that the maximum dimension of each candidate design does not exceed a certain value,

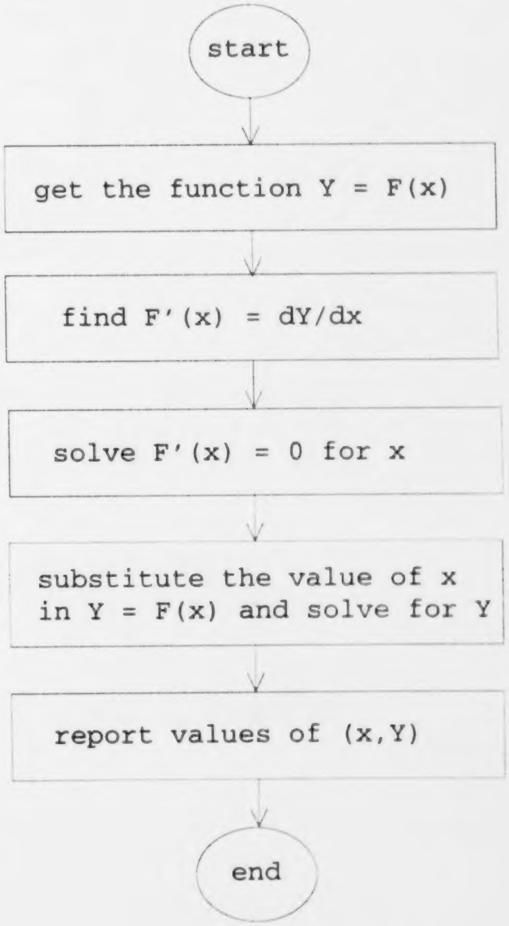

start
get the function Y = F(x)
find F'(x) = dY/dx
solve F'(x) = 0 for x
substitute the value of x
in Y = F(x) and solve for Y
report values of (x,Y)
end

Figure 4-15: A simple *procedure*

say 75 cm. Displayed in the figure is the part that deals with the dimensions in x-direction. Objects are characterized by their length, width and height. To maintain the generality of the rules, these dimensions are referred to as *x-dimension*, *y-dimension* and *z-dimension* depending on the orientation of the component.

As shown in figure 4-12, the knowledge sources of the *conceptual designer* are divided into two groups. The first group, namely that of *generating* knowledge-sources, consists of four modules: the *nomination* module, the *exploration* module, the *evaluation* module and the *verification* module. These modules are in charge of generating a number (≥ 1) of *feasible candidate solutions*, where a "feasible candidate solution" is defined as one that meets all the specifications/requirements either initially given by the user or suggested by the system in the course of design generation, as explained in chapter 2.

Definitions:

new_component:	Component to be added to an existing partial design
parent_component:	Last component in the partial design, to which the new_component is to be connected
D^x_{NC} :	Actual x-dimension of new_component
D^x_{PC} :	Actual x-dimension of parent_component
β_{NC} :	Angle between x-axis and the direction of actual x-dimension of new_component
β_{PC} :	Angle between x-axis and the direction of actual x-dimension of parent_component
D^x_O :	Maximum current dimension of the design in x direction
D^x_N :	Maximum new dimension of the design in x direction

Rules:

Rule 1: if in_series_with (new_component, parent_component) then
$D^x_N = D^x_O + D^x_{NC} \times \cos(\beta_{NC})$

Rule 2: if in_parallel_with (new_component, parent_component) and
$D^x_{NC} \times \cos(\beta_{NC}) > D^x_{PC} \times \cos(\beta_{PC})$ then
$D^x_N = D^x_O - D^x_{PC} \times \cos(\beta_{PC}) + D^x_{NC} \times \cos(\beta_{NC})$

Rule 3: if in_parallel_with (new_component, parent_component) and
$D^x_{NC} \times \cos(\beta_{NC}) \leq D^x_{PC} \times \cos(\beta_{PC})$ then
$D^x_N = D^x_O$

Rule 4: if $D^x_N > 75$ cm then
disqualified (new_component)

Rule 5: if $D^x_N \leq 75$ cm then
qualified (new_component)

Figure 4-16: Partial rule-base of a "size" expert module

The functions of the four generating modules can be outlined as follows.

- The nomination module is in charge of finding (within its components-database) all the components that partially/completely satisfy the functional requirements of the problem, that is, all those components that perform at least one of the desired elemental functions contained in the initial FBD of the problem (section 3.2.2).
- The exploration module forms and examines the knowledge-pool (section 3.2.3) of each of the components nominated by the nomination module. If possible, it solves the constraint-set of each nominated component and finds a number (≥ 1) of feasible instances of it. The information generated by this module is essential to the understanding of the components' functional behavior and hence to the functioning of other knowledge-sources.
- The evaluation module determines and presents the contribution of each explored component to the fulfilment of the overall requirements (section 3.2.4). This includes updating the functional representation of the partial design to which the new component is added. It also includes updating the representation of problem's requirements to reflect the "remaining" part of them.
- The verification module verifies the validity of the newly-augmented (through addition of new components) partial designs. It does that with respect to the overall constraints as well as the FBD requirements (section 3.2.6)

We referred to a design generated by the above-mentioned group of knowledge-sources as "candidate" to emphasize the point that such a design is not considered "final" unless it survives the criticism of the second group of modules, namely the "critic" modules.

Generating knowledge-sources have access only to the functional representation panel of the blackboard. The order in which they are activated to examine the contents of the panel is determined by *design manager*, the control module of the *conceptual designer*.

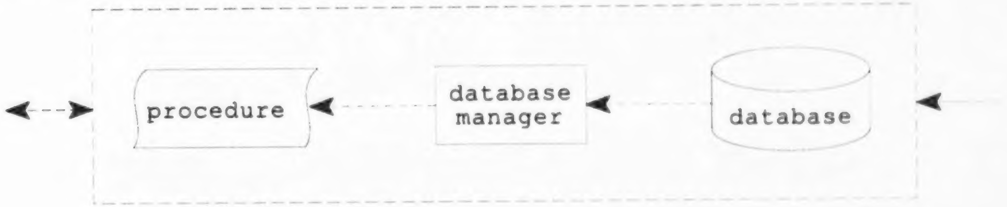


Figure 4-17: Typical organization of a *procedure* knowledge-source

Each of the four generating knowledge-sources is basically a *procedure*. The typical organization of a procedure knowledge-source is illustrated in Figure 4-17. As the figure shows, a procedure may have, associated with it, a database and a database-manager. The procedures employed by the *conceptual designer* will be elaborated in the next chapter.

The second group of the knowledge-sources, namely the *critic* knowledge-sources, can only access the structural representation panel. The order in which these modules are activated to examine the contents of the panel is also determined by the *design manager*.

Due to the nature of their functions, the critic modules are normally *rule-bases*. Each critic module represents one of the product's lifecycle objectives and uses its domain knowledge to verify/comment on the partial/complete designs produced by the *generating* modules. The typical organization of a rule-base knowledge-source is illustrated in Figure 4-18.

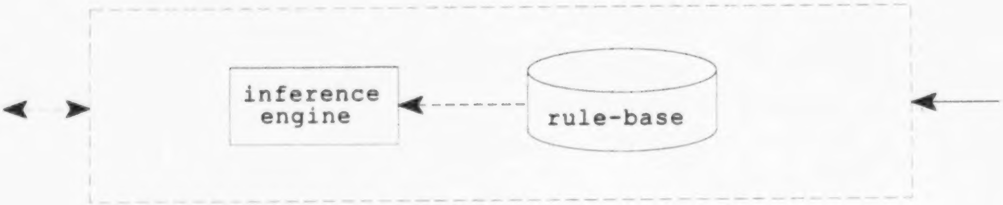


Figure 4-18: Typical organization of a *rule-base* knowledge-source

In general, the number and the contents of the critic knowledge-sources vary from application to application. While one user (individual/company/industry) may be interested in producing a cheaper, smaller and lighter product, others may wish to focus on the reliability aspects of the product. Moreover, the "knowledge" and the "mechanism" to carry out the same type of evaluation/criticism may vary from one user to another. For this reason, in this work we will not discuss the contents of the critic modules any further, although we do present a framework designed to accommodate and implement them.

4.4.3 THE CONTROL MODULE

As we pointed out earlier, the order in which each knowledge-source is activated is determined dynamically by a control module. In *conceptual designer*, this control module is called the *design manager*. Design manager continually keeps an eye on the contents of the two blackboard panels. Once the conditions for the activation of a knowledge-source are met, the design manager gives it a signal to start working on the contents of the respective blackboard panel.

Design manager is basically a rule-base where the if-part of each rule represents the activation conditions of a knowledge-source. In fact, the if-part of each rule contained in design manager is the concatenation of the if-parts of a knowledge-source's activations conditions presented as if-then rules. Figure 4-19 shows that part of the design manager which is responsible for the activation of the evaluation module.

Definitions

i: Search-tree level counter ($i = 1$: root level; $i = I$: current level)

N_i: Number of nodes in each level (N_I: number of nodes in current level)

n_i: Node counter at each level ($ni = 1, 2, ..., Ni$)

$\mathbf{C}_{ni}$: Constraint set of the new component added to (n_i)th node

$\mathbf{x}_{ni}$: Set of parameters appearing in C_{ni}

$(N\_flag)_i$: A flag indicating the completeness of the nomination stage at level i;
$(N\_flag)_i = 0$: not all the leaf-nodes at level i have been developed;
$(N\_flag)_i = 1$: all the leaf-nodes at level i have been developed

$(E\_flag)_i$: A flag indicating the completeness of the exploration stage at level i;
$(E\_flag)_i = 0$: not all the leaf-nodes at ith level have been explored;
$(E\_flag)_i = 1$: all the leaf-nodes at ith level have been explored

Rule 1:

```
if      $(N\_flag)_{I-1} = 1$          and
        $(E\_flag)_I = 1$              and
        $\forall x \in \mathbf{x}_{nI} \rightarrow$ known (x) where $n_I = 1, 2, ......., N_I$
then    activate (evaluation_module)
```

Figure 4-19: Part of the *design manager*'s rule-base

The if-part of the single rule illustrated in Figure 4-19 represents the three conditions for the activation of the evaluation module. According to the DbE model, at an arbitrary level (I) of the search tree these conditions are:

a) All the nodes at the previous (I-1) level have been developed, that is, we have augmented every partial design leading to a node at the previous level one step further (breadth-first search) to result in a new set of partial designs (level I). This is represented in the rule by the expression $((N\_flag)_{I-1} = 1)$.

b) All the new components introduced at level I (i.e. those added to the nodes at level I-1 to result in the nodes at level I) have been *explored* (chapter 3), that is, their entire functional behavior as well as design/performance parameters have been determined $((E\_flag)_I = 1)$.

c) For each partial design, values of all the parameters appearing in the corresponding constraints are determined. This is to make sure that the evaluation module *can* verify the satisfaction of the constraints. Suppose that we have N_I nodes (partial designs) at the Ith level numbered from 1 to N_I ($n_I = 1$ to N_I), and the constraint set of the newly-added component in each of these N_I designs is represented by the set ${}_{nI}\mathbf{C}$. Also suppose that the set of all variables appearing in $\mathbf{C}_{nl}$ is represented by $\mathbf{x}_{nl}$. Then the third condition can be written as the following statement.

$$\forall\, x \in \mathbf{x}_{nl} \rightarrow \text{known}\,(x)$$

Conditions for the activation of each knowledge-source are determined by the user/domain expert once at the outset of the process according to their needs/implementations techniques. This mostly applies to the *critic* knowledge-sources, as the *generating* knowledge-sources and the conditions of their activation are basically the same for all task-specific versions of *conceptual designer*. In any case, once set, activation conditions of various knowledge-sources will not be affected by the activities of the system.

4.4.4 THE USER INTERFACE

As pointed out earlier, the user interface has two basic jobs: to translate the information provided by the user to a format which would be understandable by various components of the system, and to present the final outcome of the design process to the user in a meaningful fashion.

Figure 4-20 shows the general structure of the *conceptual-designer*'s user interface. It consists of an *interpreter* and a *demonstrator*. The *interpreter* is the part in charge of expressing the "given information" (embedded in problem's function block diagram, as described in chapter 1) in the standard format of the system. Presumably, the information provided by the user falls into the following three categories.

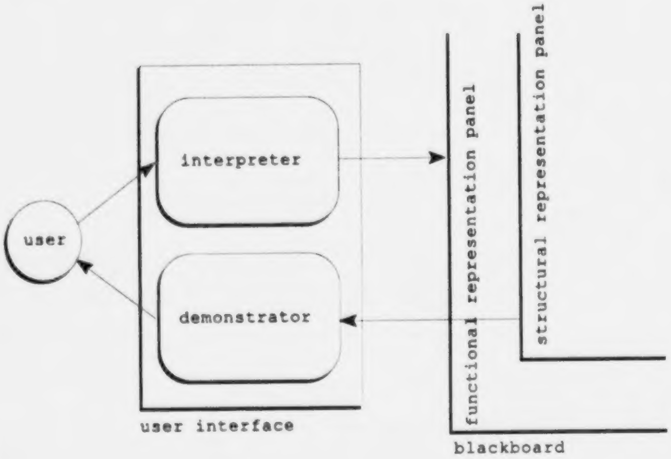

interpreter
user
demonstrator
user interface
functional representation panel
structural representation panel
blackboard

Figure 4-20: User interface: components and communications

- Equality constraints of the form $h_1(\mathbf{x}_1) = h_2(\mathbf{x}_1)$; where $\mathbf{x}_1$ is the set of all variables appearing in the equality constraints,
- Inequality constraints of the form $g_1(\mathbf{x}_2) \leq g_2(\mathbf{x}_2) \leq g_3(\mathbf{x}_2)$; where $\mathbf{x}_2$ is the set of all variables appearing in the inequality constraints,
- Qualitative information presented in predicate logic. In Prolog, the language of choice in this work, these are presented as *facts*, giving information about either a single

object or the relation between a number of objects. For example, the predicate "environment (humid)" represents the fact that the device under consideration is to be used in a humid environment (hence precautions against corrosion have to be taken regarding the choice of materials). Also, the predicates "precedes (function 1, function 2)" and "triggers (function 1, function 4)" (Figure 4-21) respectively represent the facts that in the function-block-diagram presentation of the problem, function 2 must be performed immediately after function 1, and that function 1 is a prerequisite for function 4.

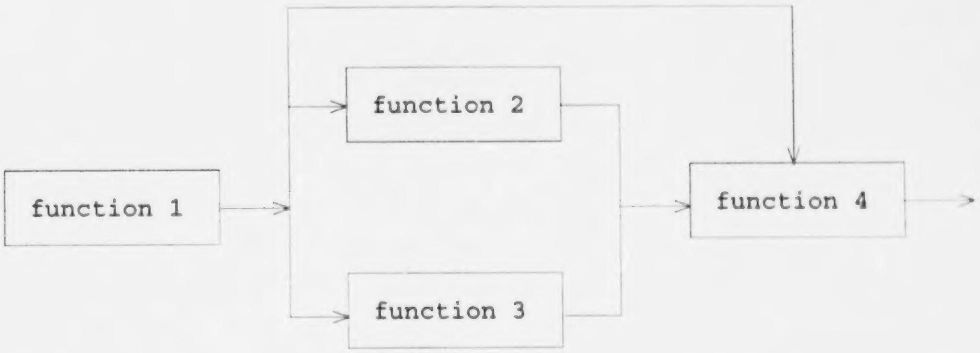


Figure 4-21: Sample FBD of initial requirements

Without the loss of generality, we further assume that the equality and inequality constraints are composed of *algebraic, trigonometric* and *exponential* functions in $\mathbf{x}_1$ or $\mathbf{x}_2$.

The user interface is basically a Prolog program which "reads" the information typed in by the user and rewrites them in the proper formats as described below.

a. *Equality constraints*

The user-interface rewrites all equality constraints, if not already, in the form $h(\mathbf{x}_1) = 0$. As illustrated in Figure 4-22, it first spots the location of the equal sign and then sequentially takes each term on the RHS to the other side while changing its sign. The important point here is to realize which signs should be changed and which ones should be kept. To facilitate this, the user is asked to use parentheses to help the system distinguish between terms and subterms. Signs of the terms will then be changed by the system but those of the subterms will not.

b. *Inequality constraints*

All inequality constraints are to be rewritten in the form $g(\mathbf{x}_2) \leq 0$. To do so, the user interface takes the following steps.

- Break up any bracketed constraint of the form $g_1(\mathbf{x}_2) \le g_2(\mathbf{x}_2) \le g_3(\mathbf{x}_2)$ (or $g_1(\mathbf{x}_2) \ge g_2(\mathbf{x}_2) \ge g_3(\mathbf{x}_2)$) into two separate constraints of the forms $g_1(\mathbf{x}_2) \le g_2(\mathbf{x}_2)$ and $g_2(\mathbf{x}_2) \le g_3(\mathbf{x}_2)$ (or $g_1(\mathbf{x}_2) \ge g_2(\mathbf{x}_2)$ and $g_2(\mathbf{x}_2) \ge g_3(\mathbf{x}_2)$, respectively).
- Rewrites each of the above as $g(\mathbf{x}_2) \le 0$ using an algorithm similar to the one applied to equality constraints (Figure 4-22) except that for the ($\ge$) case, at the end of the process the signs of all LHS terms are changed to convert the ($\ge$) constraints into ($\le$) ones.

c. *Qualitative information*

These are not altered by the user interface as the system uses the same format (predicate logic) to present its qualitative information.

The second component of the user interface, the *demonstrator,* is in charge of presenting the final product(s) of the design process and/or any terminal messages to the user. We use the term "terminal" to emphasize the point that, according to the DbE model, the user is generally not required to get involved in design activities and that he/she is provided with the outcome (success/failure) only at the *end* of the process.

Final designs are presented to the user graphically. Each component/subsystem is represented by its icon (retrieved from an icon-library) and carries a number which refers to the corresponding item in a *specifications list* associated with the diagram (Figure 4-23).

The specifications list provides the general specifications of all the components of a device. These are basically the type of information needed for rough cost estimations and do not necessarily include the detailed information required for manufacturing purposes. Catalog number of a bearing, power/speed of an electric motor and material/number of coils/mean diameter/wire diameter of a coil spring are examples of these specifications.

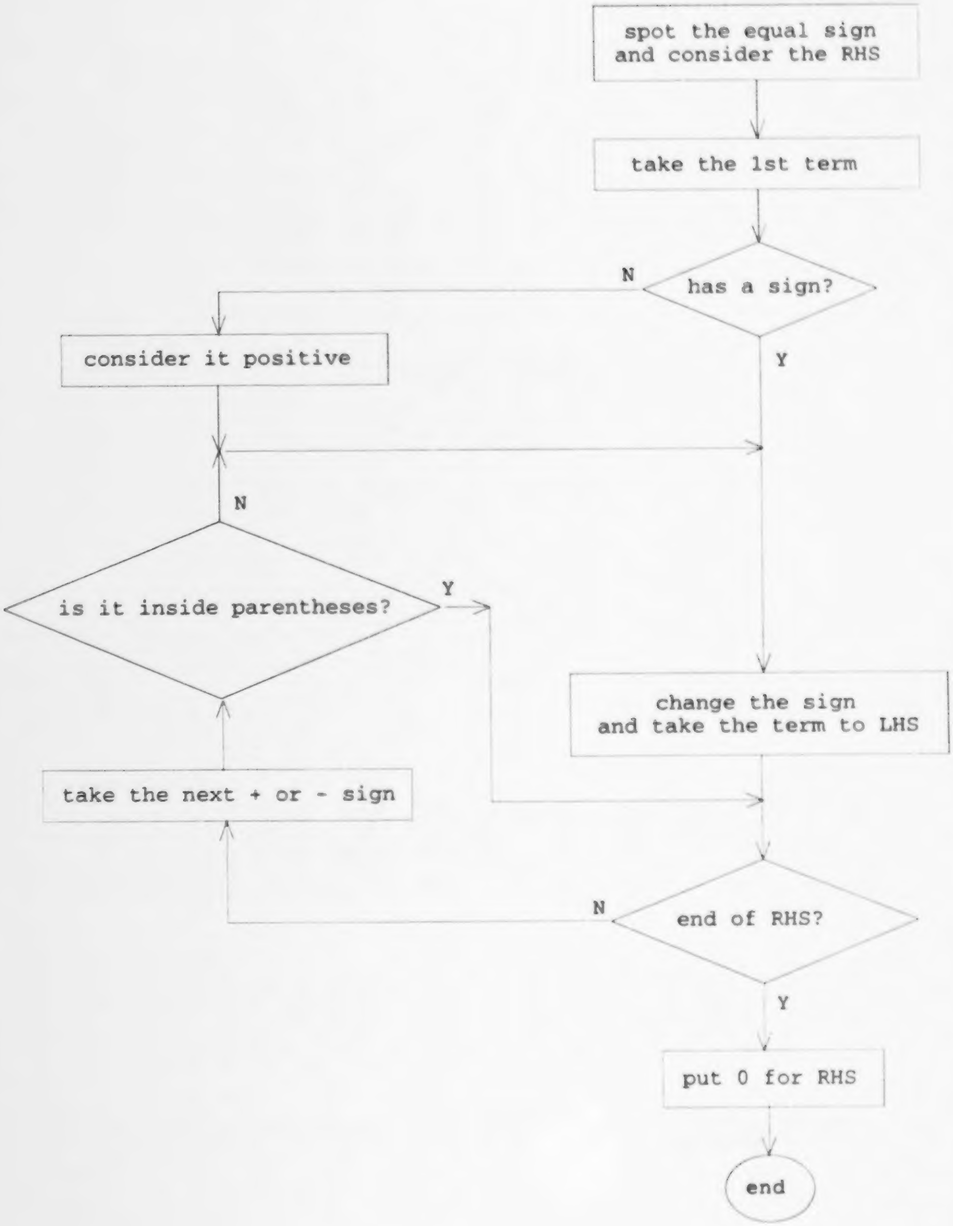
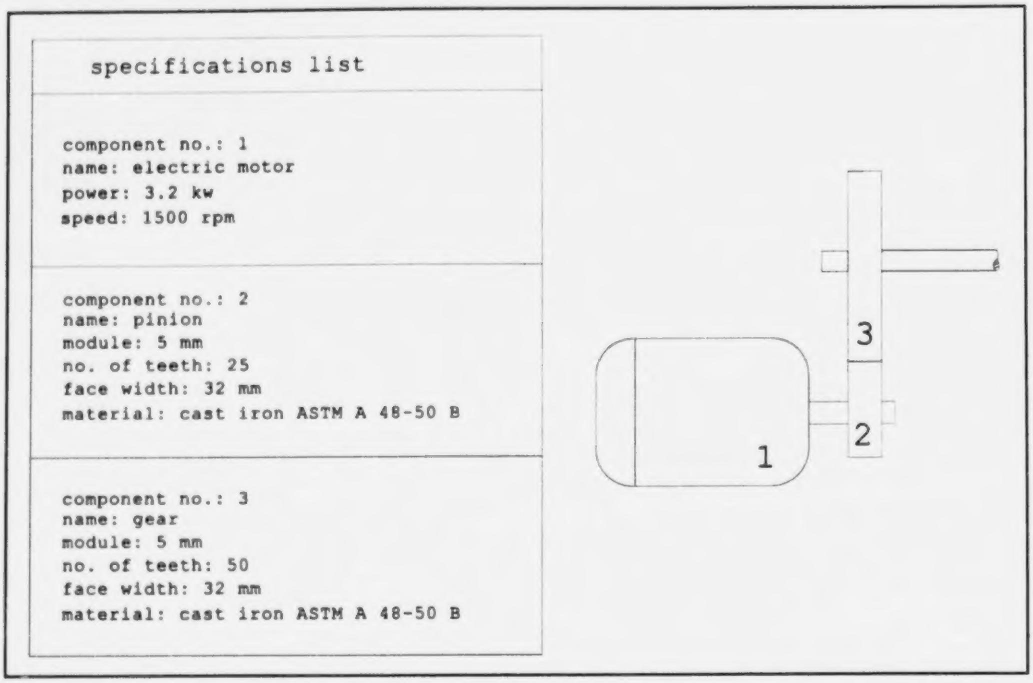

spot the equal sign and consider the RHS
take the 1st term
has a sign?
N
Y
consider it positive
N
is it inside parentheses?
Y
change the sign and take the term to LHS
take the next + or - sign
N
end of RHS?
Y
put 0 for RHS
end

Figure 4-22: How the user interface reformats equality constraints



specifications list

component no.: 1
name: electric motor
power: 3.2 kw
speed: 1500 rpm

component no.: 2
name: pinion
module: 5 mm
no. of teeth: 25
face width: 32 mm
material: cast iron ASTM A 48-50 B

component no.: 3
name: gear
module: 5 mm
no. of teeth: 50
face width: 32 mm
material: cast iron ASTM A 48-50 B

Figure 4-23: Sample graphical presentation of a design

4.5 *CONCEPTUAL DESIGNER:* THE ACTION CYCLE

Having described the architecture of the *conceptual designer* and looked at its components in some detail, we shall now provide an overview of the system *in action*, i.e. explain the order in which various components of the system take action and the way in which they interact and communicate with each other. In doing so, we shall re-visit the familiar "drill design" example from chapter 3 and follow the same design steps as before, only this time around we shall also explain who is doing what, i.e. which component of the system is carrying out which design step. To help this cause, wherever required we shall refer to the respective section/figure where we have presented the description or the results of a design step.

Our presentation of the functioning of the system in this section, however, does not include the description of the procedures whereby each knowledge-source carries out its duties. These will be elaborated in the next chapter.

The process begins with the user giving a description of the problem (Figure 3-2). In Prolog, this description is composed of a set of predicates stating the order and interrelations of *function blocks (Standard Elemental Functions)*, plus the corresponding qualitative/quantitative information (section 4.4.4). From there the system proceeds as follows.

1) The *user interface* (the *interpreter* part) reformats the given information and posts them on the functional representation panel of the blackboard and sets both "N_flag" and "E_flag" (see definition in Figure 4-19) to zero.
2) The *design manager* starts the "design generation" stage (section 4.4.1) by activating the *nomination* knowledge-source (section 4.4.2). The module searches its components-database and finds all promising components (section 3.2.2). A list of the names of these components (Figure 3-3) is reported to the functional representation panel where a search tree is set up with the component- names as its leaf nodes.
3) Once all eligible components have been found (i.e. no more components could be found to perform at least the first required elemental-function), the design manager sets the flag "N_flag" to 1 and activates the *evaluation KS* (knowledge-source).
4) The evaluation KS attempts to update the functional representation of the leaf-nodes (one is shown in Figure 4-14). As pointed out earlier, such a representation comprises an "accomplished" part and a "remaining" part for each node. Since no functional activities were previously posted at the nodes, the KS puts the initial requirements (Figure 3-2) for the "remaining" parts of all the nodes and leaves their "accomplished" parts blank.
5) Once the evaluation KS is finished with all the nodes, the design manager signals to the *exploration* knowledge-source (Figure 4-12) to access the components-list on the functional representation panel and start working on it.
6) For each node, the KS reads the name of the component and its "remaining" requirements (in the form of a set of predicates plus related information). It then forms a *knowledge-pool* (section 3.2.3) for the component. In addition to the requirements brought in from the blackboard, the knowledge-pool contains component-specific knowledge (design equations, internal constraints, data) from the components-database (same as the one accessed by the nomination database, see item 2 above).
7) The exploration KS finds a number (≥ 1) of feasible instances of the component (Figure 3-4) or, if none can be found, issues a component rejection message. Each feasible instance is posted on the functional representation panel as a new node in the search tree (Figure 3-8). In case of a rejection message, the message is as well posted on the panel for a later consideration.
8) Each time a component from the component-list on the functional representation panel is "explored" by the exploration module and the outcome is reported back to the panel,

the design manager checks to see if any components are still unexplored. If so, the exploration KS is re-activated to explore them. Otherwise (i.e. if no unexplored components are found) it sets the flag "E_flag" (Figure 4-19) to 1 and activates the evaluation KS again.

9) The evaluation module updates the functional representation of each node (section 3.2.4). It does that by comparing the initial requirements with the functional behavior of the component-instance as just reported by the exploration module. The "updated" representation of each node will then reflect the new "accomplished" and "remaining" parts (Figures 3-5 and 3-6) of the initial requirements. Those nodes whose corresponding components have been rejected by the exploration module are removed from the tree.

10) Next the design manager invokes the *verification* knowledge-source (Figure 4-12) to check the validity of each leaf-node against the FBD requirements as well as any constraints possibly generated during the exploration stage. Again, those nodes that violate either of the two requirements are removed from the search tree. The verified nodes now represent the "feasible, candidate partial-solutions" introduced in section 4.4.2.

11) Once all the leaf-nodes of the search tree have been updated and checked, the design manager calls upon the *translator layer* of the blackboard (Figure 4-12) to post the "structural equivalent" of the current search tree (Figures 3-9 and 3-13) on the structural representation panel (Figure 4-13). Both "N_flag" and "E_flag" are then set back to zero.

12) The contents of the structural representation panel now present a physical description of all "candidate" partial designs (section 4.4.2) found by the system to this point. These candidates are now to face the possible criticism of the "critic" knowledge-sources. Based on its "activation conditions", each critic KS may or may not be activated at this stage. For example, if the activation condition of a "cost" KS is that the designs be completed so that the KS can evaluate their total cost, then this knowledge-source will not be activated at a middle stage when only "partial designs" exist. Those critic modules that *are* activated, examine all the leaf-nodes of the search tree as posted on the structural representation panel, and endorse, reject or modify them. This stage was not applied to the drill example of chapter 3.

13) The "surviving" leaf-nodes of the search tree on the structural representation panel now represent the *feasible* partial/complete solutions. Complete designs are now presented to the user by the user-interface (section 4.4.4). Unless the user finds them sufficient and decides to terminate the process, the design manager re-activates the translator layer to inform the functional representation panel of any possible changes

in the search tree (by the critic modules).

14) The system proceeds by repeating steps 2 to 13 above for the partial designs on the blackboard. The cycle is repeated until one of the following cases is faced.
   - At some stage, all the leaf-nodes of the search tree represent complete designs and the tree cannot be further developed.
   - The remaining leaf-nodes (i.e. the ones not reported to the user and removed from the blackboard) violate the convergence conditions of the system (section 3.3), checked by the design manager. This means that the remaining partial designs are trapped in "loops" and the system will not converge to a solution for them.. There may also be other termination conditions such as exceeding a certain processing time or number of components.
   - Although neither of the above cases is true, the user decides to terminate the process as he/she finds the suggested complete designs satisfactory.

Figure 3-14 displays a case where all the leaf-nodes of the search tree for the drill example represent complete, feasible designs (prior to the application of critic modules).

4.6 SUMMARY

The term *system-architecture* refers to the organization of a computer system designed to perform a specific task. In this chapter we described the architecture of *conceptual designer,* a computer system to perform the conceptual design of mechanical systems as prescribed by the Design-by-Exploration model. Since *conceptual designer* has been developed according to the *blackboard* model of problem-solving, we also outlined the general characteristics of the model and its basic components as well as a special implementation of it used in conceptual designer.

A simple design problem was employed to demonstrate the course of action of the conceptual designer. We explained how each component of the system participates in the complex design-process and how it interacts and communicates with its fellow components.

Our decision to adopt a blackboard framework for our system was based on the following observations.

a) An implementation of the DbE model requires the use of multiple, diverse areas of expertise. Application of various lifecycle objectives relies on the use of an spectrum of problem-solving methods that span from exact/approximate computational methods to AI-based reasoning techniques. A blackboard architecture provides just the proper tools for accommodating such a wide variety of knowledge-types (Ensor and Gabbe 1988).

b) Though tempting at the first look, we did not find the *Object Oriented* paradigm an efficient choice for our implementation purposes. According to the paradigm, each design alternative would be considered an "object" or a collection of information represented by a set of variables plus a set of functions to operate on those variables and consequently on the object itself. Each object would be an instance of a "class" or a template for a group of objects with common characteristics. Objects "evolve" through passing messages to each other that would activate their internal functions and change their information contents.

If applied to the DbE model, the paradigm will not result in an efficient system for the following two reasons, among others. Firstly, the DbE model heavily relies on the generation and examination of multiple, diverse design alternatives which hardly fall into the same category or "class", meaning that not only "objects", but also "classes" will be continually added/removed at run-time at an unaffordable rate. Secondly, it is not practically possible to accumulate immensely diverse design-information/design-functions of various mechanical components in each object or even each class, so that the objects can "inherit" them, as prescribed by the paradigm.

c) A very attractive characteristic of the blackboard paradigm, commonly referred to as *concurrency,* allows different parts of a problem to be processed in parallel. This is highly desirable in the context of *conceptual designer,* as the independent knowledge-sources of the system can simultaneously work on different design alternatives to speed up the process . It was this characteristic that fuelled the research on the implementation of the model on parallel processors (Jones, Millington and Ross 1988).

d) A number of successful applications of the blackboard paradigm to engineering design have been reported in literature. A review of these works shows that the paradigm is capable of properly addressing many of our concerns in this research. Among the reported works is the work of Finger et al. on *concurrent design* (Finger et al. 1992) and that of Kitzmiller and Jagannathan on *automated design of air-cylinders* (Kitzmiller and jagannathan 1989).

The next chapter will contain our discussion of the procedures and techniques employed by the conceptual designer. That will conclude our presentation of the system and our approach to the automation of mechanical conceptual design process. It will also shed more light on the dark details of the rather complex architecture of *conceptual designer.*

CHAPTER 5

PROBLEM-SOLVING TECHNIQUES

In the heart of the *conceptual designer* lie the *generating knowledge-sources* introduced in chapter 4. These modules collectively play the role of the system's engine which propels it in its quest for final solution(s). Most of the system's information-processing (computations/reasoning) activities are represented by procedures and are carried out by the generating KSs. These activities vary in complexity and scope, and range from managing databases to solving complex constraint sets.

In this chapter we shall take a closer look at the procedures involved in the implementation of the DbE model. Each procedure will be described in a separate section under the knowledge-source it occupies (Chapter 4). Depending on the nature of each procedure and the complexity/novelty of the techniques it employs, we shall elaborate the basic steps of the procedure and illustrate them, wherever necessary, using simple examples. Also, in cases where more novel methods have been used, we shall provide a brief overview of the methods for the non-expert reader.

5.1 THE *NOMINATION* KNOWLEDGE-SOURCE

As explained in the last chapter, this module is in charge of finding and "nominating" all those components in a preloaded components-database which at least partially satisfy the functional requirements of the problem.. Upon receiving a query from *design manager,* the module looks for all those items in its database whose functional definitions contain the queried elemental function. Such an item is called a *candidate,* and a list of all candidates is posted on the blackboard for further references by other knowledge-sources, as explained in Chapter 4.

The organization and data-flow of the *nomination KS* is shown in Figure 5-1. The module comprises a *components library,* an *index* to this library and a *library manager.*

The components-library is basically a collection of files each representing a *component-cell.* A component-cell is a small knowledge-base which contains design-specific knowledge (data, equations, constraints) about that particular component. The contents of a typical component-cell are shown in Figure 5-2 (from Gieck and Gieck 1990; shigley 1986). These basically consist of the following.

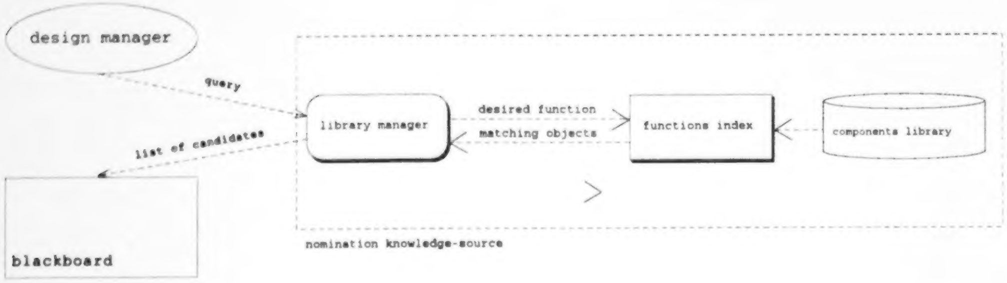


Figure 5-1: The *nomination* knowledge-source

- The name of the component,
- Its Function Block Diagram (chapter 1) giving a functional description of the component,
- A list of the Standard Elemental Functions (chapter 1) it potentially performs,
- A list of the component's design/performance parameters, their *status* and their *share functions* (if applicable). The notions of "status" and "share-function" of a parameter will be explained and used in the next section.
- Domains (ranges of values) of the design variables. These are the variables in which initial requirements of the problem are expressed. Normally, a subset of the design variables is specified and the rest are to be calculated through solving the design equations. As will be explained in the next section, these parameter-domains are needed by the system in its attempt to solve the constraint set of the problem. Note that of these domains, some are continuous while others are discrete.
- The equality and inequality constraint sets. These include the component's design equations as well as any other applicable constraints. It is mostly through these constraints that any distinctive information about the component is expressed. For example, the inequality constraint "$16 . ML \geq FW \geq 9 . ML$" in Figure 5-2 indicates that the gear face-width is normally recommended to lie between 9 times and 16 times the gear's module.
- The initial incidence matrix. The matrix indicates the presence/absence of various variables in equality constraints.

Each component-cell is stored as a separate file of the form "component_name.obj", e.g. "spur_gear_drive.obj" and "helical_compression_spring.obj". Turbo Prolog has a special utility that handles the entire collection of ".obj" files as a unit. It also allows the following operations to be carried out on the components library.

- Add "objects" to, or remove them from an existing library,
- Replace "objects" from a library,
- Extract "objects" from a library as requested,
- Sort and list the contents of a library.

In order to speed up the search for candidate components in the components-library, the nomination knowledge-source is equipped with an *index* to the contents of the library. The index is in turn a database in which each record corresponds to a component-cell (.obj file). The general format of such a record is as follows.

component (Component Name, Corresponding .obj File Address, [List of Functions])

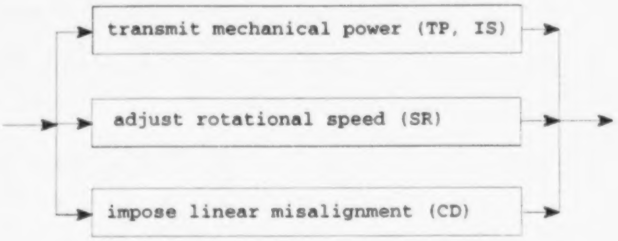
For example, the record pertaining to the spur gear drive would look like this:

component (spur_gear_drive, spr_gear.obj, [transmit_power(rot), transmit_torque(rot), adjust_rot_speed])

Each time the knowledge-source is triggered to provide information on a function, it accesses the index and sequentially checks the records for a match between their "List of Functions" part and the given function. If a match is found, the KS records the corresponding component-name and .obj file address. Once the library is completely searched, the nomination KS reports to the blackboard a list of the candidate components and the corresponding file-names. This information will be exploited by the exploration knowledge-source, as we shall see later. The operation of the nomination module is presented via its flow diagram in Figure 5-3.

Component Name: **Spur Gear Drive**

Function Block Diagram:


transmit mechanical power (TP, IS)
adjust rotational speed (SR)
impose linear misalignment (CD)

List of Functions: (Transmit_Mechanical_Power, Adjust_Rotational_Speed, Impose_Linear_Misalignment)

Design/Performance Parameters:

Parameter	*Symbol*	*Status*	*Share Function*
Speed Ratio	SR	adj	$SR_t = SR_1 \times SR_2$
Input Speed	IS	non-adj	
Output Speed	OS	non-adj	
Transmitted Power	TP	non-adj	
Center Distance	CD	adj	$CD_t = [CD_1{}^2 + CD_2{}^2 - 2 \times CD_1 \times CD_2 \times \cos(CD_1, CD_2)]^{1/2}$
Face Width	FW	non-adj	
Number of Pinion Teeth	NPT	non-adj	
Number of Gear Teeth	NGT	non-adj	
Module	ML	non-adj	
Applied Torque	AT	non-adj	
Transmitted Load	TL	non-adj	
Maximum Bending Stress	MBS	non-adj	
Allowable Stress	AlS	non-adj	
Lewis Form Factor	Y	non-adj	
Velocity Factor	K_v	non-adj	

Figure 5-2: Contents of a typical *component cell*

Parameter Domains:

SR: [0.05 to 20.00],

CD: [14 to 1200 mm],

FW: [10 to 80 mm],

NPT: [15, 16, 17, 18, 19, 20, 21, 22, 24, 26, 28, 30, 34, 38, 40],

NGT: [17, 18, 20, 24, 30, 36, 45, 51, 60, 75, 90, 100, 114, 140, 160, 180, 200, 240, 280],

ML: [1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 16.0, 20.0, 25 mm],

AIS: [20.0, 50.0, 55.0, 57.5, 72.5, 125.0 MPa]

Equality Constraints:

$$\text{eqn1} \quad AT = \frac{TP}{IS}$$

$$\text{eqn2} \quad CD = ML \times NPT \times \frac{1+SR}{2}$$

$$\text{eqn3} \quad TL = \frac{TP \times 60}{ML \times NPT \times IS \times \pi}$$

$$\text{eqn4} \quad OS = IS \,/\, SR$$

$$\text{eqn5} \quad SR = \frac{NGT}{NPT}$$

$$\text{eqn6} \quad MBS = \frac{TL}{K_v \times FW \times ML \times Y}$$

$$\text{eqn7} \quad K_v = \frac{360}{ML \times NPT \times IS \times \pi + 360}$$

$$\text{eqn8} \quad Y = -\,1.8 \times 10^{-8} \times NPT^4 + 4.6 \times 10^{-6} \times NPT^3 - 4.5 \times 10^{-4} \times NPT^2 + 0.02 \times NPT + 0.05$$

Figure 5-2 (Continued)

Inequality Constraints:

Ineq. 1 $8 \geq SR > 0$

Ineq. 2 $16 . ML \geq FW \geq 9 . ML$

Ineq. 3 $AIS \geq MBS$

Initial Incidence Matrix:

equation [eqn1, (TL, 0), (AT, 1), (MBS, 0), (FW, 0), (CD, 0), (OS, 0), (SR, 0), (IS, 1), (TP, 1), (Y, 0), (K_v, 0), (ML, 0), (NGT, 0), (NPT, 1)

equation [eqn2, (TL, 0), (AT, 0), (MBS, 0), (FW, 0), (CD, 1), (OS, 0), (SR, 1), (IS, 0), (TP, 0), (Y, 0), (K_v, 0), (ML, 1), (NGT, 0), (NPT, 1)

equation [eqn3, (TL, 1), (AT, 0), (MBS, 0), (FW, 0), (CD, 0), (OS, 0), (SR, 0), (IS, 1), (TP, 1), (Y, 0), (K_v, 0), (ML, 1), (NGT, 0), (NPT, 1)

equation [eqn4, (TL, 0), (AT, 0), (MBS, 0), (FW, 0), (CD, 0), (OS, 1), (SR, 1), (IS, 1), (TP, 0), (Y, 0), (K_v, 0), (ML, 0), (NGT, 0), (NPT, 0)

equation [eqn5, (TL, 0), (AT, 0), (MBS, 0), (FW, 0), (CD, 0), (OS, 0), (SR, 1), (IS, 1), (TP, 1), (Y, 0), (K_v, 0), (ML, 0), (NGT, 1), (NPT, 1)

equation [eqn6, (TL, 1), (AT, 0), (MBS, 1), (FW, 1), (CD, 0), (OS, 0), (SR, 0), (IS, 0), (TP, 0), (Y, 1), (K_v, 1), (ML, 0), (NGT, 0), (NPT, 0)

equation [eqn7, (TL, 0), (AT, 0), (MBS, 0), (FW, 0), (CD, 0), (OS, 0), (SR, 0), (IS, 1), (TP, 0), (Y, 0), (K_v, 1), (ML, 1), (NGT, 0), (NPT, 1)

equation [eqn8, (TL, 0), (AT, 0), (MBS, 0), (FW, 0), (CD, 0), (OS, 0), (SR, 0), (IS, 0), (TP, 0), (Y, 1), (K_v, 0), (ML, 0), (NGT, 0), (NPT, 1)

Figure 5-2 (concluded)

5.2 THE *EXPLORATION* KNOWLEDGE-SOURCE

Once the list of "candidate components", i.e. those potentially capable of performing a specific, desired function, has been posted on the blackboard by the *nomination KS*, the

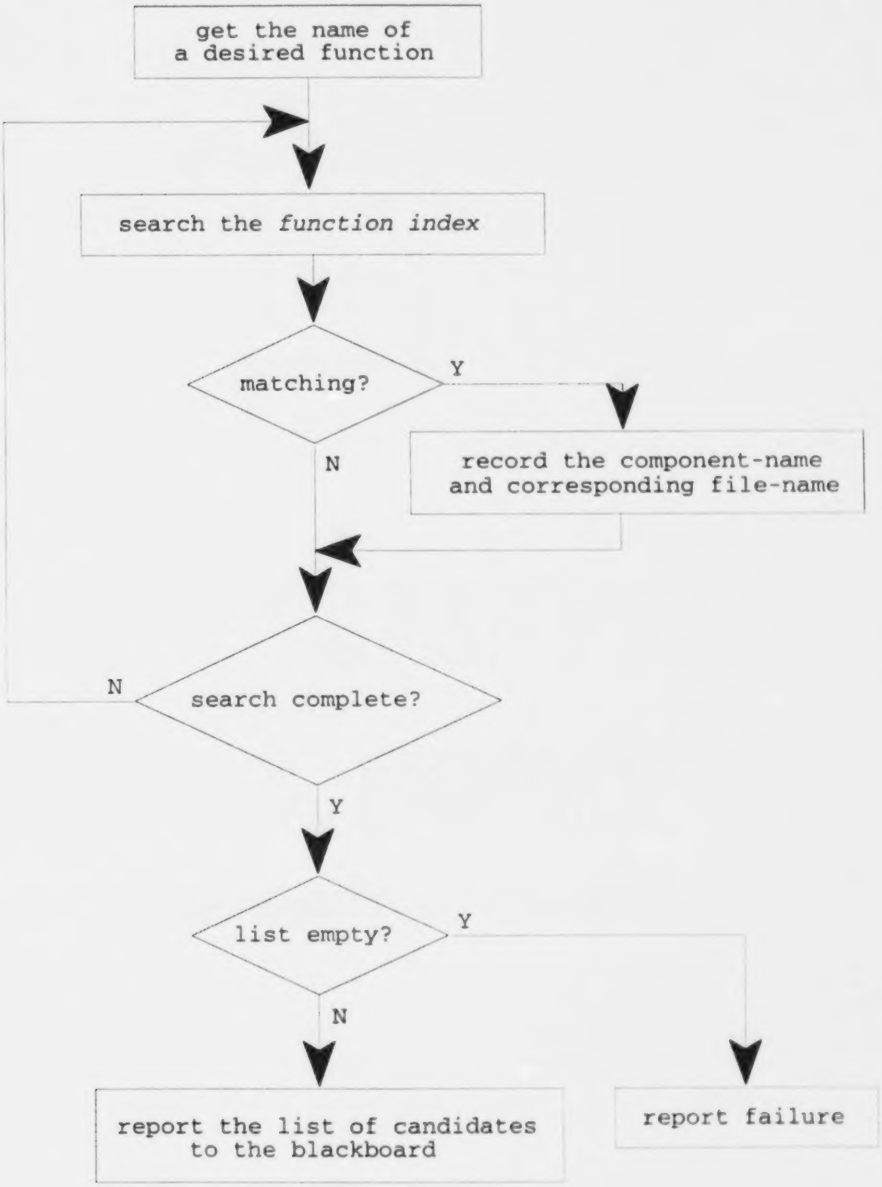

get the name of a desired function
search the function index
matching?
Y
N
record the component-name and corresponding file-name
N
search complete?
Y
list empty?
Y
N
report the list of candidates to the blackboard
report failure

Figure 5-3: The operation of the *nomination KS*

exploration KS is triggered by the design manager. The latter in turn "explores" the *knowledge-pool* (Chapter 3) of each listed component in order to reveal its complete functional behavior and to determine its contribution to the fulfilment of the initial requirements. The results are then reported back to the blackboard (Figure 5-4).

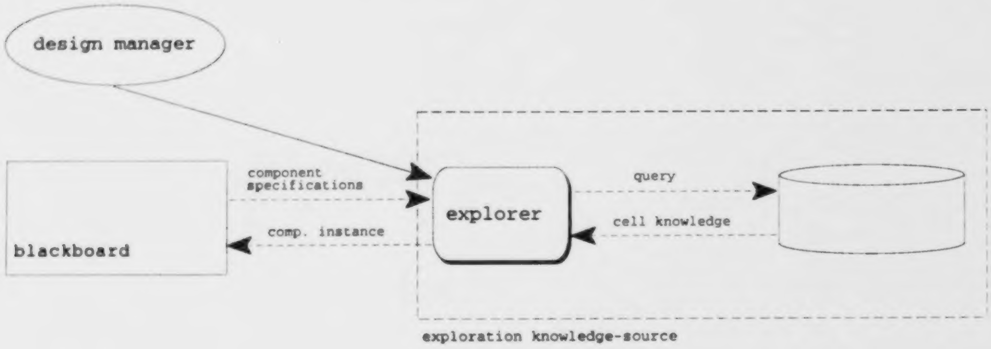


Figure 5-4: The *exploration* knowledge-source

Earlier in this work we argued that in a typical design problem the following three types of knowledge are generally present in the knowledge-pool of a component (Chapter 3).

- The element-specific knowledge contained in the respective "component-cell" (section 5.1). This includes a list of the functions the component is capable of performing, its form-function relations, and any other information on the functional behavior of the component.
- The initial information provided by the user. This basically comprises the values of a subset of component's design variables plus a set of (external) constraints representing part of the design requirements.
- Other life-cycle (e.g. manufacture, cost, reliability) considerations introduced by critic expert modules (Chapter 4) and/or the user in the form of a number of constraints.

Figure 5-5 shows the sequence of actions taken by the exploration module. Given the knowledge-pool of the component, the module will encounter the challenge of performing the following tasks for *any* given component, with minimum or no need for user-intervention (as implied by the concept of *design automation*).

1) To evaluate the knowledge-pool from an *information-consistency/redundancy* point of view to determine if the problem can actually be solved with the available knowledge,
2) To apply a "universal" solution strategy to the problem, regardless of the nature of the constraint set, and

3) To manage to find *multiple feasible solutions* to the problem, if they exist.

Of the three tasks mentioned above, some have been individually studied, completely or partially, by other researchers and reported in the literature. Nevertheless, to our knowledge they have never been addressed collectively in the context of an automated design system. As we proceed with the rest of this work, we shall refer to other researchers' contributions to the techniques we are about to present.

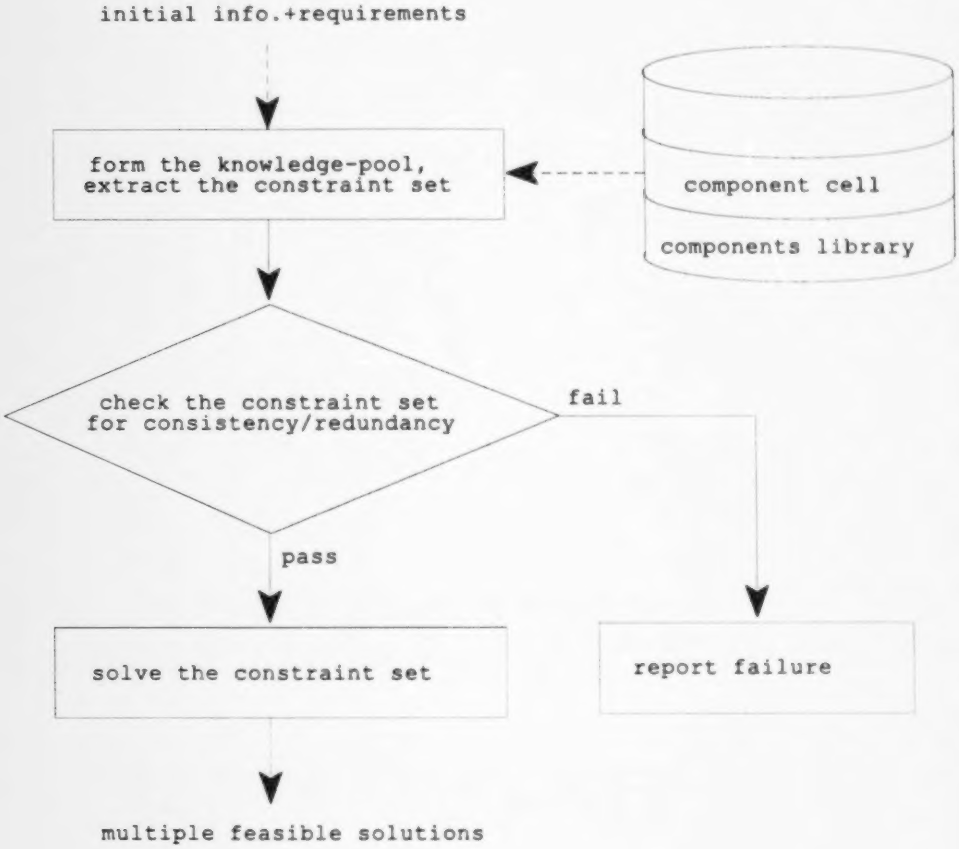

initial info.+requirements
form the knowledge-pool, extract the constraint set
component cell
components library
check the constraint set for consistency/redundancy
fail
pass
solve the constraint set
report failure
multiple feasible solutions

Figure 5-5: Operation of the *exploration KS*

5.2.1 PROBLEM FORMULATION

The problem we are dealing with in the exploration stage is generally refered to as *constraint management*. A set of constraints, representing the design-governing relations, requirements and given specifications is "extracted" from the knowledge-pool of the candidate-component. The goal is now to find solution(s) to this set, if they exist. Without loss of generality, in this chapter we assume that all extracted constraints are presented in terms of the component's design/performance parameters.

The constraint set, in its general form, is represented by $m+k$ constraints in $n+l$ variables as follows.

$\mathbf{H} = \{h_i\}$ is the set of m equality constraints of the form $h_i(\mathbf{x_1}) = 0$ $i = 1, 2, ..., m$;

and

$\mathbf{G} = \{g_j\}$ is the set of k inequality constraints of the form $g_j(\mathbf{x_2}) \leq 0$ $j = 1, 2, ..., k$

where $\mathbf{x_1} = (x_{11}, x_{12}, ..., x_{1n})$ and $\mathbf{x_2} = (x_{21}, x_{22}, ..., x_{2l})$ are the sets of variables appearing in **H** and **G** respectively, and $\mathbf{x} = \mathbf{x_1} \cup \mathbf{x_2}$ represents the set of design variables of the problem. Note that usually, though not necessarily, $\mathbf{x_1}$ and $\mathbf{x_2}$ have some members in common and $x_1 \cap \mathbf{x_2} \neq \emptyset$, that is, the total number of *distinct* variables (or $|\mathbf{x}|$) is less than $n+l$. We further represent the collective constraint set of the problem as $\mathbf{C} = \mathbf{H} \cup \mathbf{G}$.

In practice, however, the collective constraint set **C** is formed by the junction of two other constraint sets. One is the set of constraints included in the initial, user-provided information and the other is the set of constraints contained in the component cell. Each of these two sets generally has both equality and inequality constraints in it.

As mentioned earlier, the initial information included what we called the *external constraints*, plus the values of a subset of **x** introduced above. We also called the constraints contained in the component cell the *internal constraints*. Once the exploration module is activated, the first thing it does is to combine the internal and external constraints to form **C**. A *feasible instance* of the component is then represented by a set of values for **x** which satisfies **C**.

As allowed by the DbE model, the overall constraint set **C** and the initial specifications, or the knowledge-pool in short, may change at virtually any stage of the design process. New constraints may be introduced by the "critic" knowledge-sources and given specifications may be changed/augmented due to the conditions of fellow components. Each time something in the knowledge-pool is altered, the design manager re-activates the module to seek the feasible solution(s) through solving the new **C** for the new specifications.

Briefly then, the constraint management problem at hand is that of consecutively studying, and possibly solving, a (changing) constraint set **C** in a set of variables **x**, with values of a (different) subset of **x** being given at each time. The set needs to be examined before each solution is attempted, because it may or may not be solvable due to problems such as inconsistency and redundancy.

The concept of treating the entire process of component design (in our case the exploration process) as a constraint management problem is fairly new, though the general constraint management problem has a rather long history and has been addressed by many researchers. However, the majority of the works reported in the literature on constraint management are limited in scope and do not generally meet the complex requirements of *design automation* as laid out in this work. A number of constraint-based systems have nonetheless been reported in the literature that address some aspects of *automated constraint management* rather closely. Here we shall take a brief look at some of the related work.

A mathematical presentation of the constraint theory can be found in (Friedman and Leondes 1969a; 1969b; 1969c). Steward (1981), Himmelblau (1966) and Soylemez (1973) were among the first to consider the automation of constraint management in a *consistent equality constraint set.* Self-consistency and absence of inequality constraints are two of the main assumptions in their works. These assumptions, however, are not generally correct of the *conceptual designer.* First because inequality constraints are often present in the problem and second because we have no knowledge about the consistency of the constraint set *a priori.*

Based on the belief that mechanical designers should be *optimal designers* rather than just *feasible designers*[1], a number of researchers have regarded the constraint management problem in the context of mechanical design as a single-/multi-objective optimization problem (Johnson 1971 and 1980; Wilde 1978; Siddall 1982; Arora 1989; Papalambros 1987; Papalambros and Wilde 1988; Reklaitis 1983; Haug and Arora 1979; Jain and Agogino 1990). Several computer- and knowledge-based systems have been developed with the purpose of automating the process of optimal design (Chieng and Hoetzel 1987; Mistree, Hughes and Phuoc 1981; Parkinson, Balling and Free 1984; Balachandran and Gero 1987; Mehta and Korde 1988; Arora and Baenziger 1986; Li and Papalambros 1985).

Agogino and her group (Jain and Agogino 1990; Michelena and Agogino 1988) have further studied the role of qualitative analysis in reducing the complexity of the design problems before applying optimization techniques to them. Inspired by the ideas of Wilde (Wilde 1975; Wilde 1986; Papalambros and Wilde 1988) and following the works of Papalambros (Papalambros and Wilde 1979; Papalambros and Li 1983) and Azarm and Papalambros (1984) they have developed a set of computer programs (SYMON and its extension SYMFUNE) (Choy and Agogino 1986; Agogino and Almgren 1987a; 1987b) that use monotonicity analysis to verify the problem formulation and possibly "shrink" its search space. They have also developed a design methodology (called 1st PRINCE) (Cagan and Agogino 1987) which augments the results of these programs with first principle information to achieve some degree of innovation in optimal design.

---

[1] Remember that the DbE model prescribes the generation of multiple feasible designs rather than a single, optimal one (Chapter 3).

Since we believe *monotonicity analysis (MA)* has been somewhat overrated by some researchers, let us take a quick look at what the method does and what its limitations are.

MA is based on the application of the Karush-Kuhn-Tucker optimality conditions (Karush 1939; Kuhn and Tucker 1950; Kuhn 1976) and is presented in the context of constrained optimization problems. Such a problem is typically represented by an objective function $F(\mathbf{x})$ subject to a number of constraints of the general form "$f(\mathbf{x}) \leq 0$" where $\mathbf{x} = [x_1, x_2, \ldots\ldots, x_n]$ is the set of variables of the problem. Other presentations of the constraints can simply be transformed to the above form.

Monotonicity analysis is a tool for utilizing the "monotonic properties" of the objective function and the constraints to reason about their behavior. It attempts to reduce the dimensionality of the optimization problem and to detect flaws in the problem formulation before it is numerically solved. Before we outline the "rules" of MA, the following terms have to be defined.

- The *monotonicity* of a differentiable function $f(\mathbf{x})$ with respect to variable x_k is the algebraic sign of $\partial f / \partial x_k$. The function is said to be strictly monotonically increasing (decreasing) w.r.t. x_k if and only if $\partial f / \partial x_k > (<) 0$, for all $\mathbf{x}$.
- A variable x_k is said to be *bounded* from above (below) by a constraint $c(\mathbf{x}) \leq 0$ if it achieves its maximum (minimum) value at strict equality, i.e. at $c(\mathbf{x}) = 0$.
- A constraint is active (inactive) at $\mathbf{x}_0$ if $c(\mathbf{x}_0) = 0\ (< 0)$.

MA has three fundamental rules (theorems) for defining well-constrained optimization problems.

Rule one:	If the objective function is monotonic with respect to a variable, then there exists at least one active constraint which bounds the variable in the direction opposite to the objective.
Rule two:	If a variable is not contained in the objective function, then it must be either bounded from both above and below by active constraints or not actively bounded at all, that is, all constraints monotonic with respect to that variable must be inactive.
Rule three:	The number of nonredundant active constraints cannot exceed the total number of variables. In other words the *dimensionality* of an active constraint set, or the number of its degrees of freedom, cannot be negative.

Let us apply these rules to a simple example to demonstrate the kind of insight that can be gained through MA. The example and part of its discussion are taken from.

$$
\begin{array}{ll}
\text{Maximize} & f = 3x_1 + 2x_2 \\
\text{subject to} & g_1 = 5x_1 + 4x_2 - 23.7 \le 0 \\
 & g_2 = -x_1 + 2x_2 - 4 \le 0 \\
 & g_3 = -x_3 - 1 \le 0 \\
 & h_1 = x_3 - 2x_1 - 2 = 0
\end{array}
$$

All the three design variables (x1, x2 and x3) are assumed to be *positive integers.*

Discussion:

- First we may apply *rule 3* above to x_3 as it does not appear in the objective function. According to the rule, x_3 should be either bounded from both above and below by active constraints or not bounded by nonredundant active constraints at all. Since h_1 is always active (= 0 at optimum), we may re-write it as ($h_1 = x_3 - 2x_1 - 2 \le 0$) and consider g_3 as active ($g_3 = 0$ at optimum). This would bound x3 from both above and below. However, it will give x_3 a value of (-1) which is not acceptable. Therefore g_3 cannot be active and may be modified as ($g_3 = -x_3 - 1 < 0$). By the same token h_1 will be redundant (will not play a role in finding x_1 and x_2).
- We now apply rule 1 to x_1. Since the objective function is monotonically increasing w.r.t. x_1, then there must be an active constraint to bound it in the opposite direction, i.e. from above. "g_1" happens to do just that, so we consider it active (= 0). This, however, will violate the condition of the variables being *integers,* and is rejected. Again, g_1 is re-written as ($g_1 = 5x_1 + 4x_2 - 23.7 < 0$).

At this stage, this is all monotonicity analysis could tell us: that the problem should be restated as follows.

$$
\begin{array}{ll}
\text{Maximize} & f = 3x_1 + 2x_2 \\
\text{subject to} & g_1 = 5x_1 + 4x_2 - 23.7 < 0 \\
 & g_2 = -x_1 + 2x_2 - 4 \le 0
\end{array}
$$

In effect, the above analysis has reduced the dimensionality of the problem by one degree of freedom and the total number of constraints by two.

Now suppose that g_1 was originally given as ($g_1 = 5x_1 + 4x_2 - 23.0 \le 0$). Taking the same steps as before and applying the first rule to g_1, now we would be able to consider it active ($g_1 = 5x_1 + 4x_2 - 23.0 = 0$), as it would no longer violate the "integer" condition. The latter would yield: $x_1 = -0.8x_2 + 4.6$ and we would be able to re-write the objective function as ($f = -0.4x_2 + 13.8$) and g_2 as ($g_2 = 2.8x_2 - 8.6 \le 0$) or ($x_2 \le 3.07$). This in turn will result in ($x_2 = 2$, $x1 = 3$ *and* $f = 13$) which is the solution. We can see that this time around, the whole problem has been solved just by reasoning about the constraints.

Despite its capability to detect many of the ill-formulated optimization problems and facilitate their solution, *monotonicity analysis* cannot be considered the ultimate, universal constraint-management tool, especially in the context of design automation the way we have introduced. Here are some of the reasons why.

- The rules of MA are necessary, but not sufficient, conditions for a well-constrained optimization problem. Consider the following simple example (Agogino and Almgren 1987b).

  Minimize $f(x_1, x_2) = x_1 / x_2$

  Subject to $g_1(x_1, x_2) = x_2 / (x_1)^2 - 1 \leq 0$

  for positive x_1 and x_2

  The problem satisfies all three MA conditions for being well-constrained. There are no variables in the constraint that do not appear in the objective function (rule 2) and, as we shall see shortly, the number of active constraints (one) does not exceed the number of variables (two). As for the first rule, the objective function is monotonically increasing w.r.t. x_1 and decreasing w.r.t. x_2. Hence rule 1 calls for g to be active in order to bound x_1 from below and $_2x$ from above. This yields $(x_1 / (x\ )^2 = 1)$. Substituting the latter in f will result in the reduced problem (Minimize $f = 1/x_1$ subject to $x_1 > 0$) which does not have a finite solution. The problem is not well-constrained, though the rules of MA say it is.

- In general, there is no guarantee as to whether all the constraints in the constraint-set of a component are continuous and differentiable as required by MA. Also, a combination of continuous and discrete variables are often present in the variable-set of the component.

- The centre of focus in monotonicity analysis and its predecessors is to route the search in an optimizing direction and seek the optimal solution through skipping not only the infeasible solutions but also the sub-optimal feasible ones. As we discussed earlier (chapter 3), this is in contrast with the requirements of DbE because a sub-optimal component instance may yet lead to an optimal system design. For the reasons previously discussed (chapter 3) we do not consider the constraint-management problem at hand an *optimization* problem. We would rather seek to find all *feasible* instances of a candidate component than a single, optimal one.

- Though relatively successful in handling complex problems with numerous constraints, MA does not show the same prospect in case of the DbE scenario, where numerous problems of low- to medium complexity are to be rapidly, and sometimes repeatedly, evaluated. The cumulative computations and reasoning operations of MA will noticeably slow down the design process and yet will not provide enough help to justify its use.

The argument just presented does by no means deny the necessity of a design problem being properly constrained. On the contrary, we believe that such an analysis is required to avoid fruitless computations on poorly-formulated problems that are likely not to have a finite solution. Our approach to handling the constraint-management problem will be presented in the following section[2].

Qualitative analysis of constraints with the purpose of gaining insight into the constraint management (and not necessarily design optimization) has also been addressed by Kannapan and Marshek (1991), Rinderle (Rinderle and Suh 1982; Rinderle and Watton 1987), Ishii and Barken (1987a; 1987b) and Agrawal et al. (Agrawal et al. 1993) among others. The scope of these works ranges from simply suggesting the order in which design equations should be solved (while almost ignoring the inequality constraints) to providing tools for assessing the "goodness" of a design according to the axioms of good design (Suh 1990).

Favoring *feasible* design rather than *optimal* design, Sridhar et al. (Sridhar, Agrawal and Kinzel 1991) propose a stepwise transformation of inequality constraints of the problem to equality ones through introducing *slack variables.* Considering the equality constraints and using the occurence matrix of the problem, they check the inequality constraints one by one and transform only those whose activation will help reduce the dimensionality of the problem. This is as opposed to the classical optimization approach wherein *all* inequality constraints are transformed to equations. The Minimum Design Deviation Algorithm they propose requires the user to occasionally free some of the bound variables.

Serrano and Gosssard (1987; 1988) and Serrano (1984; 1990) have investigated the problem of constraint management in the context of mechanical conceptual and concurrent design. Their constraint-based system MATHPAK (Serrano 1984), and its extension the "Concept Modeller" (Serrano and Gossard 1987) use a graph theoretical approach to the evaluation of constraint sets and the detection of conflicts/redundancies in those sets.

Given the constraint set of a component, they use the equality constraints along with a number of activated inequality constraints (Section 5.2.1) to generate a *design point* in the solution space, which is basically the equivalent of a candidate design. The rest of the inequality constraints are then checked for violations. If violations are detected, one or more of the following actions are taken accordingly.

- Values of one or more variables are changed so that the whole constraint set is satisfied,

---

[2] Classical techniques to verify the problem formulation are relatively well established (see references in the text, especially Papalambros and Wilde 1988). For the sake of brevity, in this chapter we restrict our discussion of the system evaluation to those areas in which we are presenting more novel techniques, and assume that the problem is properly bounded otherwise.

- Some/all of the activated inequality constraints are replaced so that the solution of substituting active constraints plus the equality constraints result in parameter values that no longer violate the constraint set,
- If neither of the above works, the problem is ill-formulated. Therefore, the problem formulation (some/all of the constraints) is changed.

In all of these cases the proposed system prompts the user to intervene and provide the necessary values / perform the necessary modifications. For the very same reason, this strategy too, fails to meet all requirements of automated design.

With this brief background in mind, we now present our approach to handling the constraint management problem within the context of the DbE model of conceptual design. But first we make the following observations.

- The constraint set of a typical candidate component comprises both equality and inequality constraints in the component's design/performance parameters.
- The constraint set is initially consistent. Both the contents of the component-cells and the initial requirements presented by the user are carefully set up and are initially conflict-free. Inconsistencies *may* occur later in the process when new constraints are introduced.
- In its original form (prior to the *activation* of any of the inequalities) the problem could be overconstrained, underconstrained or exactly constrained, depending on the initial specifications. This can only be determined through examining the set of equality constraints.
- If overconstrained (i.e. the number of equality constraints exceeding that of the unknown variables by k), the problem cannot be solved unless k equations are proved to be dependent on others (and hence redundant).
- If underconstrained (i.e. the number of unknown variables exceeding that of the equality constraints by r), the problem will not have a unique solution and is said to have r degrees of freedom. Also, nonlinear problems can have non-unique solutions.
- In case of an underconstrained problem with r degrees of freedom, specifying the values of r unknown variables *may* render the set exactly constrained.
- A feasible solution to the problem is represented by a set of parameter values which satisfies the collective constraint set, i.e. both the equality and the inequality constraints.

5.2.2 PROBLEM EVALUATION

The problem-solving strategy of the exploration module is based on a *Genetic Search* (to be elaborated in Section 5.2.5) of the solution space to spot the feasible points representing feasible solutions. The space to be searched is *r*-dimensional , where *r* is the problem's degrees of freedom defined as the number of variables less the number of independent equality (active) constraints.

Recognition of independent equality constraints requires an examination of the equality constraint set. Hence as the first step the exploration module scans **C**, the collective constraint set of the problem, and forms the two disjoint subsets of equality and inequality constraints (**H** and **G** respectively). An examination of **H** will now determine the *output variables* of the equality subset. This in turn will serve in specifying the solution subspace to be searched. An *output variable* is the choice of dependent variable in an equality constraint for which the constraint is to be solved, with other variables being determined by other relations, or guessed.

We take **H** to represent the set of *m* equality constraints in *n* variables (x_{11}, x_{12}, ..., x_{1n}). Depending on the nature of the constraints and the initial specifications **H** could be underconstrained, exactly constrained (constraint bound) or overconstrained. In practice, however, the underconstrained case dominates and, especially in the context of DbE, we often get more unknown variables than equality constraints.

This situation may be considered favorable by some in the sense that it gives the designer more room (more degrees of freedom) to maneuver. It could also provide for more alternative solutions as an initially-unbound variable would theoretically take a multitude of values from across its range, instead of one prescribed value, and it is likely that more than one of these values qualify and thus result in multiple solutions. As we shall discuss shortly, DbE takes full advantage of this phenomenon in its quest for multiple design alternatives.

Having distinguished the equality subset, we now employ the notion of *maximal matching* to evaluate the subset. By *matching* we mean assigning one output variable to each constraint. A matching **M** is represented by a set of ordered pairs, of which the first member is an unknown variable and the second is an independent equality constraint. In formal representation $\mathbf{M} = \{(x_1 , h) : x_1 \in \mathbf{x_1} , h \in \mathbf{H}\}$, where $\mathbf{x_1}$ is the set of *n* variables appearing in **H**. **M** represents a one-to-one mapping of the variable set onto the equality set, that is, no two pairs in **M** may have their first and/or second member in common.

Maximal matching refers to the largest possible M. The maximal matching is said to be complete if $|\mathbf{x_1}| = |\mathbf{H}|$ and no element remains unmatched in either$_1$ $\mathbf{x}$ or **H**. It is important to note that equal number of elements in $\mathbf{x_1}$ and **H** does not necessarily indicate a complete maximal matching as there might still be some unmatched constraints or variables.

If a number of constraints end up unmatched then the system is *overconstrained* (*m-n* more constraints than unknowns) unless the extra constraints are redundant, that is, they are consistent with the rest of the set and are satisfied by the values of the unknown variables determined by the rest of the equations. Except in this circumstance, a unique solution cannot generally be found for an overconstrained system.

In case of an overconstrained set the exploration module considers the possibility of completing the maximal matching by "freeing" some of the initially specified variables and calculating and suggesting new values for them. We shall further discuss the case later on in this section.

On the other hand, if a number of variables end up unmatched then the system is *underconstrained* (*n-m* more unknowns than constraints). At first glance it might seem that in this case the specification of values of any *n-m* unknown variables will complete the matching and render the system exactly constrained (a case we need for solving the equation set). This is, however, not generally true because these *n-m* values might render the set overconstrained due to the presence of redundant constraints as frequently happens in real world problems.

Consider, for example, the following set of three equations in four unknowns represented by its *incidence matrix.* (The incidence matrix $[a_{ij}]$ of a set of p equations in q variables is a $p \times q$ matrix in which $a_{ij} = 1$ if variable j appears in equation i, and $q_j = 0$ otherwise.)

$$\begin{array}{lcccc} & (\mathrm{x1}) & (\mathrm{x2}) & (\mathrm{x3}) & (\mathrm{x4}) \\ (\mathrm{eqn.}\ 1) & 1 & 1 & 0 & 0 \\ (\mathrm{eqn.}\ 2) & 1 & 1 & 0 & 0 \\ (\mathrm{eqn.}\ 3) & 1 & 1 & 1 & 1 \end{array}$$

The system is underconstrained ($m = 3$ and $n = 4$) with one degree of freedom and we need to specify the value of one variable to make it exactly constrained. If we specify the third or fourth variable, then the set will be exactly constrained and a complete maximal matching can be found. (For example if we specify x3, then x1 can be assigned to eqn.1, x2 to eqn. 2 and x4 to eqn. 3). On the other hand, if we specify the first or second variable, we shall end up with an overconstrained set (say we specify x2, then we shall have equations 1 and 2 in x1 and equation 3 in x1, x3 and x4).

Therefore in the case of an underconstrained system we need to know not only the number of unmatched variables but also their combination. Having recognized the unmatched variables, the overall degrees of freedom of the problem can be determined by simply adding the number of unmatched variables to the number of variables that appear in the inequality set **G** and not the equality set **H**. In formal representation

$$\mathrm{DOF} = (|\mathbf{x}_1| - |\mathbf{H}|) + (|\mathbf{x}_2| - |\mathbf{x}_1 \cap \mathbf{x}_2|). \qquad (5\text{-}1)$$

[Note that the RHS of equation (5-1) can simply be rewritten as ($|\mathbf{x}|$ - $|\mathbf{H}|$), i.e. *the total number of variables less the number of equality (active) constraints.*]

Steward (1981) proposes an algorithm for finding the maximal matching in an equation set. A modified and enhanced version of his algorithm (Figure 5-6) is employed by the exploration module (EM) to carry out the analysis just discussed. We shall discuss these enhancements shortly. EM starts out with the initial incidence matrix of the equality set (excluding the equations representing the initial specifications) and marks the specified variables with "S" *(Specified)* for future references. It then tries to sequentially match each unassigned variable with an unassigned equation in which the variable appears.

If all the equations are successfully assigned, EM marks those variables that are neither specified nor assigned with "U", indicating an *Unmatched* variable. Removing the columns associated with the "U" variables results in a square incidence matrix representing a maximal matching and an exactly constrained equality subset.

Steward's algorithm basically stops here and does not consider further processing of the incidence matrix in the case where the successful assignment of all equations is not possible. In an extention to the original algorithm, EM then re-examines the set to see if manipulating the initial specifications can fix the set and find a maximal matching.

To re-examine the set, EM unmarks the "S" variables one at a time and treats them as free (unbound) unknowns. For each "S" variable unmarked, its matching equation (row) is removed from the matrix and the assignment procedure just described is re-performed. This goes on until either a maximal matching is reached or no "S" variables are left. The success of this tactic would mean that the initial specifications had made a subset of equations overconstrained and that we have managed to render the subset exactly constrained by freeing some of its bound (specified) variables and letting them be calculated along with other unknowns.

If, after freeing all the bound variables, the procedure still fails, we may infer that the overconstrained subset cannot be fixed and a solution for the equality set can not be generally found. In this case the exploration module reports failure, meaning that no instance of the candidate component could be found to meet the initial requirements.

At the end of the assignment algorithm, EM scans the inequality set **G** to find those elements of $\mathbf{x}_2$ that are neither part of $\mathbf{x}_1$ nor initially specified. It then adds these variables to its list of unmatched variables from **H** to form what we shall call the set of *free variables*. The product of the assignment algorithm is a list of unmatched variables plus the output assignments in **H** (represented by the modified incidence matrix).

Let us illustrate the operations presented so far in this section by applying them to a classical gear design problem[3]. Although for this purpose we would only need to know the equality constraints of the example problem, we shall present the complete problem here so that we could pursue our solution steps in the following sections and complete the solution. We have chosen the standard spur-gear design problem to make it easier for the reader to compare the performance of the algorithms presented here with that of the other systems which have been applied to similar problems (including those reported in Jain and Agogino 1990; Langrana, Mitchell and Ramachandran 1986; Gabriele and Serrano 1991; Zarefar, Lawley and Etesami 1986; Ramachandran, Langrana and Steinberg 1990).

We wish to design a spur-gear drive that transmits 10 kw of power at an (input) speed of 500 rpm while reducing the speed by a factor of 4 (output speed = 125rpm). The drive is part of a transmission system in which the centre distance of the input and output shafts is to lie between 0.2m and 0.4m.

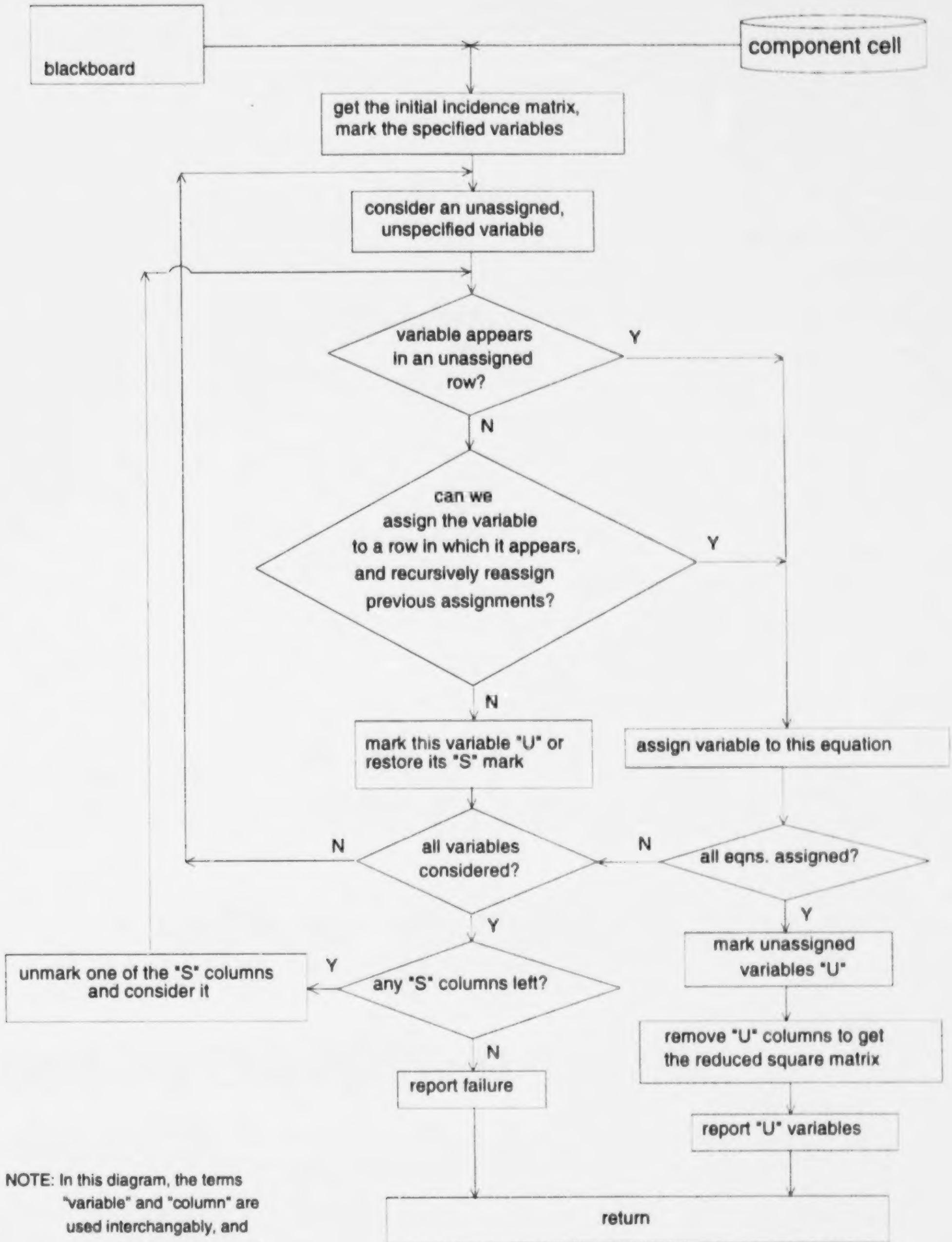
Here is a list of the variables ($\boldsymbol{x}$) used in the presentation of the problem.

SR : Speed Ratio (Input to Output)
IS : Input Speed(rpm)
OS : Output Speed(rpm)
TP : Transmitted Power(w)
NPT : Number of Pinion Teeth
NGT : Number of Gear Teeth
ML : Module(m)[4]
FW : Face Width(m)
CD : Centre Distance(m)
AT : Applied Torque(N-m)
TL : Transmitted Load(N)
MBS : Maximum Bending Stress(Pa)
AlS : Allowable Stress(Pa)
K_v : Lewis Form Factor
Y : Velocity Factor

---

[3] In the *classical gear design problem* commonly discussed in design textbooks, only the algebraic relations representing primary functions are considered . The differential equations representing the secondary functions such as internal vibrations are not taken into account.

[4] Note that the term *module* has been used in two different meanings in this text: a *program module* is an independent piece of code with a specific function; whereas *gear-module* is defined as the ratio of the pitch diameter and the number of teeth in an involute gear.


blackboard
component cell
get the initial incidence matrix, mark the specified variables
consider an unassigned, unspecified variable
variable appears in an unassigned row?
Y
N
can we assign the variable to a row in which it appears, and recursively reassign previous assignments?
Y
N
mark this variable "U" or restore its "S" mark
assign variable to this equation
N
all variables considered?
N
all eqns. assigned?
Y
Y
unmark one of the "S" columns and consider it
Y
any "S" columns left?
mark unassigned variables "U"
N
remove "U" columns to get the reduced square matrix
report failure
report "U" variables
return

NOTE: In this diagram, the terms "variable" and "column" are used interchangably, and so are the terms "equation" and "row".

Figure 5-6: The *assignment* algorithm

Figure 5-7 presents the problem's *internal constraints* (design equations plus inequality constraints) from the respective component set (Figure 5-2).

The requirements of the problem can be summarized as follows (Figure 5-8). Note that we have intentionally included an incorrect specification (OS = 200 rpm, instead of 500 / 4 = 125 rpm) to help better verify the performance of the algorithm.

$$(1) \qquad AT = \frac{TP}{IS}$$

$$(2) \qquad CD = ML \times NPT \times \frac{1+SR}{2}$$

$$(3) \qquad TL = \frac{TP \times 60}{ML \times NPT \times IS \times \pi}$$

$$(4) \qquad OS = IS \,/\, SR$$

$$(5) \qquad SR = \frac{NGT}{NPT}$$

$$(6) \qquad MBS = \frac{TL}{k_v \times FW \times ML \times Y}$$

$$(7) \qquad k_v = \frac{360}{ML \times NPT \times IS \times \pi + 360}$$

$$(8) \qquad Y = -1.8 \times 10^{-8} \times NPT^4 + 4.6 \times 10^{-6} \times NPT^3 - 4.5 \times 10^{-4} \times NPT^2 + 0.02 \times NPT + 0.05$$

$$(9) \qquad AIS \geq MBS$$

$$(10) \qquad 8.0 \geq SR > 0$$

$$(11) \qquad 16 \times ML \geq FW \geq 9 \times ML$$

Figure 5-7: Internal constraints of the spur-gear drive

The overall constraint set (**C**) of the problem, extracted from its knowledge-pool, is made up of the constraint relations in Figures 5-7 and 5-8.

(1) TP = 10,000 w,
(2) IS = 500 rpm,
(3) SR = 4.0,
(4) OS = 200 rpm
(5) 0.4m ≥ CD ≥ 0.2m

Figure 5-8: Initial requirements of the gear design problem

To illustrate the performance of the *assignment* algorithm, we need to distinguish the equality subset (**H**). In this case **H** comprises constraints (1) to (8) in Figure 5-7 plus constraints (1) to (4) in Figure 5-8. The incidence-matrix representation of the equality subset is shown in Figure 5-9. Note that according to the algorithm, the initial-specification relations (constraints (1) to (4) in Figure 5-8) are not to be included in the matrix. Equation numbers refer to those in Figure 5-7.

In the incidence matrix of Figure 5-9, bound variables (columns) have been marked with "S" for *specified*. In this case these represent the equality constraints of Figure 5-8. Also in the matrix we have replaced the "1"s with "x"s and "0"s with "."s for convenience and clarity.

						S	S	S	S					
	TL	AT	MBS	FW	CD	OS	SR	IS	TP	Y	K_v	ML	NGT	NPT
eq.1	.	x	.	.	.	.	.	x	x	.	.	.	.	.
eq.2	.	.	.	.	x	.	x	.	.	.	.	x	.	x
eq.3	x	.	.	.	.	.	.	x	x	.	.	x	.	x
eq.4	.	.	.	.	.	x	x	x	.	.	.	.	.	.
eq.5	.	.	.	.	.	.	x	.	.	.	.	.	x	x
eq.6	x	.	x	x	.	.	.	.	.	x	x	x	.	.
eq.7	.	.	.	.	.	.	.	x	.	.	x	x	.	x
eq.8	.	.	.	.	.	.	.	.	.	x	.	.	.	x

Figure 5-9: Initial incidence matrix of the spur-gear design equations

As the matrix shows, the set comprises 8 equations in 14 variables of which 4 have been initially specified. That leaves us with 8 equations and10 unknowns. It looks like by specifying two more variables we would be able to render the set constraint-bound and possibly solve it. Let us now apply the assignment algorithm (Figure 5-6) and see if this prediction is true and if so, which two variables should be specified. The algorithm works as follows.

Starting with the first row, variable "AT" is found both unassigned and unspecified, so it is assigned to equation 1, meaning that the equation is to be solved for "AT". Moving on to the second row, "CD" is found to have the same qualifications and hence is assigned to equation 2. Similarly, "TL" is assigned to equation 3. In row 4, all three variables are specified, meaning that there is no unknown variable left to be calculated from equation 4.

There is no need to proceed further to realize that a maximal matching cannot be found under current circumstances. This is because the number of variables of the problem is greater than that of the equations, so the "largest possible" matching (section 5.2.2) would have a dimension equal to the number of equations, in this case 8. Now since we are losing one equation and hence one assignment, we will not be able to have a matching of size 8 (a maximal matching) and possibly a solution.

To overcome the impass, the exploration module "frees" the first specified (**S**) variable "OS". The situation is now re-examined. This time, variable "OS" is assigned to equation 4 with no changes in previous assignments. Continuing with equation 5 we find variable "NGT" unassigned and unspecified. "NGT" is therefore assigned to equation 5. Similarly, "MBS" is assigned to equation 6, "k_v" to equation 7 and "Y" to equation 8.

As for the remaining variables, "FW" does not appear in an unassigned row and if we assigned it (instead of "MBS") to equation 6, we would have the same problem with "MBS" and would not be able to re-assign it to another equation. Hence "FW" is labelled "U" for "Unmatched". Also, variables "ML" and "NGT" only appear in already-assigned equations and if we assigned them to equations they appear in, two other variables would lose their assignments and the situation would not improve. Therefore these two variables are labled "U" as well.

Let us now take a closer look at what happened in row 4 of the matrix. There we had the equation $(OS = IS / SR)$ with all the three variables specified (in this particular case with inconsistent values, as $200\ rpm \neq 500\ rpm / 4$). This would obviously deny the constraint set a maximal matching as there would be no assignments for this equation, meaning that the equation would not be used to calculate the value of any unknowns.

As we explained earlier, the situation proclaims that the initial specifications had made a subset of equations overconstrained. In this case the overconstrained subset is equation 4 (one equation, zero unknown) or more correctly, the following set of four equations in three unknowns.

(1) $OS = IS / SR$

(2) $IS = 500$ *rpm,*

(3) $SR = 4.0,$

(4) $OS = 200$ *rpm*

Using the assignment algorithm, we have managed to render the subset exactly constrained by freeing one of its bound (specified) variables (OS) and letting it be calculated along with other unknowns. This is like ignoring equation (4) above to render first the remaining subset and then the original set itself exactly constrained.

One should note that in the assignment algorithm we are primarily concerned with the constraint set being well-constrained rather than with the values of the variables. For instance in the example above, we would not care if the initial specifications were or were not consistent.

The product of the assignment algorithm is shown in Figure 5-10. It is basically the same incidence matrix in which now a pair of parentheses indicates the assignment of a variable to an equation and "**U**" labels tag the unmatched (free) variables.

				U			S	S	S			U		U
	TL	AT	MBS	FW	CD	OS	SR	IS	TP	Y	K_v	ML	NGT	NPT
eq.1	.	(x)	.	.	.	.	.	x	x	.	.	.	.	.
eq.2	.	.	.	.	(x)	.	x	.	.	.	.	x	.	x
eq.3	(x)	.	.	.	.	.	.	x	x	.	.	x	.	x
eq.4	.	.	.	.	.	(x)	x	x	.	.	.	.	.	.
eq.5	.	.	.	.	.	.	x	.	.	.	.	.	(x)	x
eq.6	x	.	(x)	x	.	.	.	.	.	x	x	x	.	.
eq.7	.	.	.	.	.	.	.	x	.	.	(x)	x	.	x
eq.8	.	.	.	.	.	.	.	.	.	(x)	.	.	.	x

Figure 5-10: Incidence matrix; assignments and free variables

The *matching* (M) is therefore defined as follows.

M = {(AT, eq.1), (CD, eq.2), (TL, eq.3), (OS, eq.4), (NGT, eq.5), (MBS, eq.6), (K_v, eq.7), (Y, eq.8)}

This gives us one possible set of free variables: "FW", "ML" and "NPT". To these we add a fourth variable "AlS" (Allowable Stress) which appears only in the inequality set and is not already specified. The problem therefore has four degrees of freedom [equation (5-1) for $|\mathbf{x}_1|$

= 14, $|\mathbf{H}| = 11$ (8 design equations plus three initial specifications), $|\mathbf{x}_2| = 6$ and $|\mathbf{x}_1 \cap \mathbf{x}_2| = 5$]. By removing the "U" and "S" columns from the matrix of Figure 5-10 we get a square incidence matrix which represents the exactly-constrained version of the equality set (Figure 5-11).

Note that the reduced matrix of Figure 5-11 is not necessarily unique. The proposed algorithm could have come up with a different arrangement if it had started with a different row (i.e. if the rows in the original incidence matrix had been re-arranged).

	TL	AT	MBS	CD	OS	Y	K_v	NGT
eq.1	.	(x)	.	.	.	.	.	.
eq.2	.	.	.	(x)	.	.	.	.
eq.3	(x)	.	.	.	.	.	.	.
eq.4	.	.	.	.	(x)	.	.	.
eq.5	.	.	.	.	.	.	.	(x)
eq.6	x	.	(x)	.	.	x	x	.
eq.7	.	.	.	.	.	.	(x)	.
eq.8 .	.	.	.	.	(x)	.	.	

Figure 5-11: The reduced (square) incidence matrix of the spur gear drive

To summarize, in this section we learned how the exploration module examines the equality subset and determines:

a) whether the equality constraints (can be made to) form an exactly-constrained system and hence can potentially be solved,

b) which variables can the equations be solved for and which equation will calculate which variable,

c) which variables are "extra", i.e. are not assigned to- and calculated by the equations. These "free" variables must be somehow *guessed* before the equality subset could be solved for the assigned variables.

5.2.3 GENERAL SOLUTION STRATEGY

Remember that our original constraint management problem was stated as follows.

> "Examine the constraint set for well-boundedness and, if possible, find the values of the variables so that they satisfy the collective constraint set, i.e. both equality and inequality constraints."

In the previous section we explained how to get the first part done, i.e. to find/render the problem well-bounded. As for the second part, we just learned that in practice we only need to specify the values of the other unknowns. The rest is simply a matter of checking the inequality constraints for satisfaction. Hence the second part of the problem can be restated as follows.

> "Find values of the free variables that, along with values of the other unknowns obtained from the solution of the equation set, satisfy the inequality constraints."

We refer to those values of the free variables that meet the above requirement as *feasible values.* For example, in the gear design problem we need to find the feasible values of variables "FW", "ML", "NPT" and "AlS". That is, we are after such values of these variables that when substituted in equations 1 to 8 of Figure 5-7 and appended to the variables thus calculated, they collectively satisfy the inequality constraints of the problem (Figure 5-12).

Like the equality constraint subset, the inequality subset (**G**) is extracted from the component's knowledge-pool by the exploration module and is presented in Figure 5-12 in the standard form $g_j(\mathbf{x}_2) \le 0$. Note that each bracketed constraint has been presented as two one-sided constraints.

$$
\begin{aligned}
CD - 0.4 &\le 0 \\
0.2 - CD &\le 0 \\
SR - 8.0 &\le 0 \qquad (SR > 0) \\
FW - 16\,ML &\le 0 \\
9\,ML - FW &\le 0 \\
MBS - AlS &\le 0
\end{aligned}
$$

Figure 5-12: The inequality constraint subset (**G**)

A trivial approach to finding the feasible values of the free variables in a problem would be to use *slack variables* to incorporate the inequality constraints into the equality subset, and then find those values of the free variables that lead to the (consistent) solution of the augmented equation set. In the gear design example, the augmented equation set will be as shown in Figure 5-13. Equations 9 to 11 represent the initial specifications and equations 12 to 17 represent the inequality constraints of Figure 5-12 turned into equations using positive slack variables (SV_i).

The other option is to leave the inequality constraints as they are and find those values of the free variables that, together with the values of other unknowns obtained from the solution of the equation set, satisfy the inequality constraints.

In both cases, the solution of the problem involves searching the **r**-dimensional space of the free variables where **r** is the number of these variables. However, the difference is that the first approach requires the solution of a larger equation set with more unknowns compared to the second approach.

For instance, in the gear design example the slack-variable approach requires the solution of 17 equations in 17 variables (excluding the free variables and considering the new unknowns SV_i) whereas in the second approach these numbers are cut down to 11 equations and 11 unknowns.

The solution strategy used by the exploration module is based on a special version of the second approach. Our re-formulation of the problem and the search technique that EM uses in its quest for multiple solutions justify the choice of this approach, as we shall explain in the rest of this section.

5.2.4 RE-FORMULATING THE PROBLEM

In order to facilitate the monitoring of the inequality set, the exploration module first combines all the inequality constraints (**G**) (Figure 5-12) into a single *penalty function* of the following form.

$$PF = \sum_{i=1}^{k} [\max\ (0,\ \overline{g_i}(\boldsymbol{x_2}))]^2 \tag{5-2}$$

where k is the number of inequality constraints in **G**;

$\boldsymbol{x_2}$ is the set of variables appearing in **G**, and

$\bar{g}$ represents the normalized form of **g**. The normalization is meant to make the penalty function equally sensitive to the variations of its constituent terms. This will be illustrated later on for the gear design example.

Now the problem can be restated one more time as to find the values of the free variables so that they satisfy the equality set (**H**) and minimize (zero) the above penalty function which is always non-negative. In this new formulation, the problem may be considered a *pseudo-optimization* problem. We define a pseudo-optimization problem as one

(1) $AT = \dfrac{TP}{IS}$

(2) $CD = ML \times NPT \times \dfrac{1+SR}{2}$

(3) $TL = \dfrac{TP \times 60}{ML \times NPT \times IS \times \pi}$

(4) $OS = IS \,/\, SR$

(5) $SR = \dfrac{NGT}{NPT}$

(6) $MBS = \dfrac{TL}{k_v \times FW \times ML \times Y}$

(7) $k_v = \dfrac{360}{ML \times NPT \times IS \times \pi + 360}$

(8) $Y = -1.8 \times 10^{-8} \times NPT^4 + 4.6 \times 10^{-6} \times NPT^3$

$- 4.5 \times 10^{-4} \times NPT^2 + 0.02 \times NPT + 0.05$

(9) $TP = 10{,}000$

(10) $IS = 500$

(11) $SR = 4.0$

(12) $CD - 0.4 + SV1 = 0$

(13) $0.2m - CD + SV2 = 0$

(14) $SR - 8.0 + SV3 = 0$

(15) $FW - 16\ ML + SV4 = 0$

(16) $9\ ML - FW + SV5 = 0$

(17) $MBS - AIS + SV6 = 0$

Figure 5-13: Augmented equation set of the spur-gear drive

in which:

- the minimum of the objective (penalty) function is already known (in this case zero),
- present in the problem are both equality constraints (the equation set) and inequality constraints (ranges of the free variables) that curb a set of variables not necessarily appearing in the objective function.

Before we present our approach to solving the above problem in its new formulation, we should point out that in practice, the penalty function just introduced is rarely, if at all, a well-behaved function in its general form. Nonlinearities in the problem and such characteristics of the penalty function as multimodality and presence of both continuous and discrete variables as well as the need for multiple simultaneous solutions defy the realm of most classical optimization techniques. Also, the proportions of the solution space in real world problems (e.g. 10^{10} points for a problem with as few as 5 free variables and 100 values per variable) make the use of enumerative search methods impractical.

5.2.5 THE SOLUTION SCHEME

Based on the arguments just presented, the exploration module uses a novel search technique called *Genetic Algorithms* (GAs) to solve the pseudo-optimization problem just stated. This decision is motivated by the following characteristics of GAs:

- In their search for the optimum of an objective function, GAs only use the value of the function itself and do not rely on other function-related information such as differentiability, monotonicity and continuity. Hence, no matter how complex and ill-behaviored the function may be, the method will handle it properly.
- GAs can work with various types of variables (e.g. numerical/non-numerical and continuous/discrete) at the same time. This makes them applicable to a broader range of real-world design problems normally containing a mix of variable-types including such non-numerical variables as the choice of a material or the catalog designation of a standard component.
- GAs simultaneously work on a group of points in the search space instead of a single point. That is, at each time they would be processing a multitude of potential solutions, and in the end they would typically result in multiple solutions. The implicit parallelism also means that, for very large search spaces, GAs are much more efficient than enumerative schemes.

Since our discussion of EM's search scheme heavily relies on GA terminology, for the less-familiar reader we present here a brief overview of the method. The implementation of the method will be then illustrated in the context of the gear design example of previous section.

5.2.5.1 GENETIC ALGORITHMS[5]

Among the stochastic direct search methods is the Genetic Algorithm, based on the principles of natural selection and survival of the fittest. The terminology used by GAs is quite similar to that used in natural genetics and even a close analogy is maintained between the elements of the two. We shall define this terminology as we go on.

The basic structure processed by GAs is the *string (chromosome).* A string is a concatenation of a number of codes (often binary codes) of given length. The string bits (0 or 1 in a binary string) are the equivalents of natural *genes.* Each individual code represents a design variable (similar to a characteristic in natural genetics such as "eye color") and each specific instance of the code represents, directly or indirectly, a specific value of that variable (equivalent of, say "blue eye"). There are as many codes in a string as the number of design variables, hence a string basically represents a possible design (solution).

Suppose, for example, that we want to determine the dimensions of a box in inches with maximum volume subject to the constraint that the total area of the box is constant. The problem has three variables L, W and H and an objective function ($V = L \times W \times H$) to be maximized subject to ($L \times W + L \times H + W \times H = 432 \text{ in}^2$). Further suppose that each of the three variables can take any integer value between 1 and 31 inches inclusive.

Now if we wanted to use a binary code to codify different values of each variable, we would need a five-digit binary code to do that ($2^5 - 1 = 31$). This way the code "00001" would represent (in this case directly) the first value in the variable's domain (i.e. 1 inch), "00010" to the second (i.e. 2 inches) and "11111" to the 31st value (i.e. 31 inches). Then the string "10010 01010 11001" would represent a candidate design (solution) specified by the 18th (10010) value of the first variable, the 10th (01010) value of the second variable, and the 25th (11001) value of the third variable, or a box of 18" × 10" × 25".

GAs start out by generating an initial *population* of strings through random selection of string-bit values. The number of strings (chromosomes) in the population is called the *population size.* The population size is initially specified by the user, or is determined by the computer according to heuristic rules (discussed later) and is usually kept constant throughout the search. In the example above, in order to randomly generate one string we would need to flip a coin (run a random binary generator) 15 times (number of bits in the string) and with a population size of, say 5, we would need to do that 75 times (5×15)[6]. An instance of the population thus generated is shown in Figure 5-14[7]. The population represents *generation* zero of the chromosomes.

---

[5] Although the basic steps of GAs have been pretty much standardized, there are often more than one way to implement them. Unless otherwise stated, in this work we have tried to stay with the interpretations and recommendations of David Goldberg (1989), which appear to be the most coherent and complete of all.

[6] This population size is chosen only for demonstration purposes. Later in this section we shall discuss how to choose the optimum population size for a certain chromosome length.

String No.	String (Chromosome)	Real Dimensions	Fitness
1	01010 11100 00100	10" × 28" × 4"	1120
2	00011 10010 10010	3" × 18" × 18"	972
3	01000 01000 10111	8" × 8" × 23"	1472
4	00100 01100 11000	4" × 12" × 24"	1152
5	00110 00111 11110	6" × 7" × 30"	1260

Figure 5-14: Typical generation zero for the box-design example

The second column in the above table contains the randomly generated 15-bit strings each representing a set of variables (dimensions) for the box. The real (decoded) values of the variables are presented in the third column. In this particular case, the decimal equivalent of each 5-bit binary code directly represents the real value of the corresponding variable. For example, the decimal equivalent of the binary code "10010" is 18 which happens to be the same as the actual 18th value in the variable's domain (18").

This may not always be the case; decimalized codes may refer to different numerical or even non-numerical values. For example, if the variable's domain comprised even integers between 2 and 62 inclusive, then the the same binary code "10010" would represent a value of 36" instead of 18", as this time the 18th value in the variable's domain would be 36. By the same token, the same code may refer to a color "purple" as the 18th color in the variable's domain.

The fourth column in the above table contains the *fitnesses* of various strings (or various candidate designs). The fitness of a string (a candidate design) is evaluated using the objective function. Since GAs seek to maximize the fitness of their candidate solutions, in a maximization problem this fitness could be simply expressed as the value of the objective function calculated for the specific parameter values each string represents, i.e. the fitness function could be the same as the objective function. This is the case with the box design example where the fitness of each candidate box is expressed simply as its volume.

However, in a minimization problem the fitness obviously decreases with the increase of the objective function. One way to compensate for this is to define the fitness function as follows.

---

[7] Note that we have rejected those values that violate the "area" constraint. The number of "coin flips" mentioned here therefore does not include the ones resulting in invalid values. Although this is not the only (or the most efficient) approach to constraint satisfaction in GAs, it is adequate for our brief overview of the method. We shall further discuss the issue later in this chapter.

Fitness Function = K - Objective Function

where K is a constant, large enough to exclude negative fitnesses. A value commonly used for K is the sum of the minimum and maximum values of the objective function in each generation.

Having analysed and evaluated the strings in generation zero, the next generation (generation one) is now created from the fittest members (strings) of generation zero. To do this, each string is given a weighting proportional to its fitness in a selection process whereby *parents* of the members of the next generation are selected. The process is called *fitness proportionate reproduction* and determines the number of copies of each string in the current generation to go to a *mating pool,* where the selected strings get a chance to participate in producing the strings of the next generation.

Let us see how fitness-proportionate reproduction works. The easiest way to do this is to simulate the process with the operation of a weighted roulette wheel. Each string in the population has a wheel slot, sized in proportion to its fitness. If the we let f'_i represent the *raw fitness*[8] (Figure 5-14, column 4) of the ith string in the population and $\sum f'$ represent the sum of the raw fitnesses of all strings in the same population, then the *relative fitness* (and hence the *probability of selection*) of the ith string would be $f'_i / \sum f'$. Figure 5-15 shows relative fitnesses of the population of Figure 5-15 in percentage form.

String No.	String (Chromosome)	Raw Fitness	Relative Fitness
1	01010 11100 00100	1120	18.7%
2	00011 10010 10010	972	16.3%
3	01000 01000 10111	1472	24.6%
4	00100 01100 11000	1152	19.3%
5	00110 00111 11110	1260	21.1%
Total		5976	100.0%
Average		1195.2	

Figure 5-15: Raw and relative fitnesses of the population of Figure 5-14

---

[8] The term *raw fitness* here indicates in this brief "tutorial" we are not using a *scaling* technique to improve the reproduction probabilities of the fitter strings and depress those of the weaker ones, as is a common practice in GAs applications. Nevertheless, EM's solution scheme does employ such a technique and we shall discuss it later on in this work.

The corresponding roulette wheel is shown in Figure 5-16.

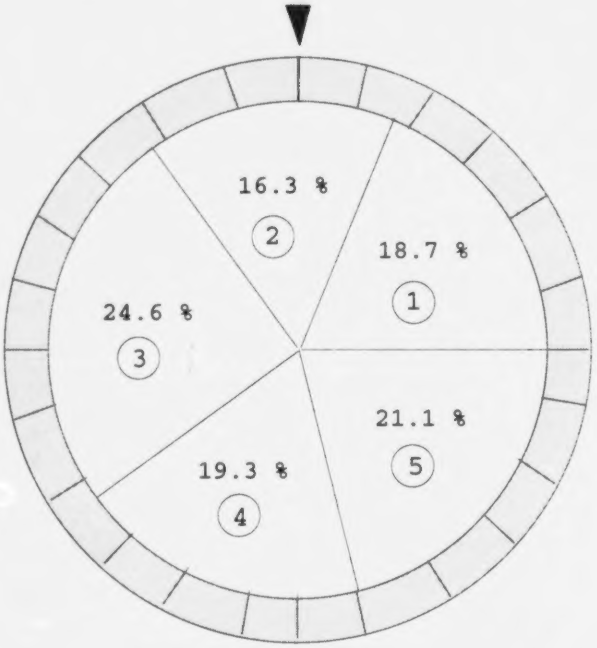


Figure 5-16: Weighted roulette wheel for generation zero

To *reproduce*, we simply "spin" the roulette wheel as many times as the population size, in this case 5 times. Each spin will reveal a "winning" number (in this case between 1 and 5), which specifies the string whose copy will make it to the next stage, i.e. the *mating pool*. It is evident that the larger the wheel-slot of a string (i.e. the fitter the string), the higher its probability of having copies in the mating pool and thus participating in the creation of the next generation.

Let us suppose that in a typical sequence of spins of the wheel in Figure 5-16, strings 1, 4 and 5 are each selected once, string 3 is selected twice and string 2 is not selected at all. The resulting mating pool is shown in Figure 5-17.

String No. (Mating Pool)	String No. (Generation Zero)	Selected String	Real Dimensions	Fitness
1	1	01010 11100 00100	10" × 28" × 4"	1120
2	3	01000 01000 10111	8" × 8" × 23"	1472
3	3	01000 01000 10111	8" × 8" × 23"	1472
4	4	00100 01100 11000	4" × 12" × 24"	1152
5	5	00110 00111 11110	6" × 7" × 30"	1260

Figure 5-17: *Mating pool* for generation zero

The sequence of actions on a typical population is graphically shown in Figure 5-18. The strings in the mating pool *mate* at random, that is, pairs of strings are randomly selected, mixed and possibly altered by *genetic operators* to produce strings of the succeeding generation.

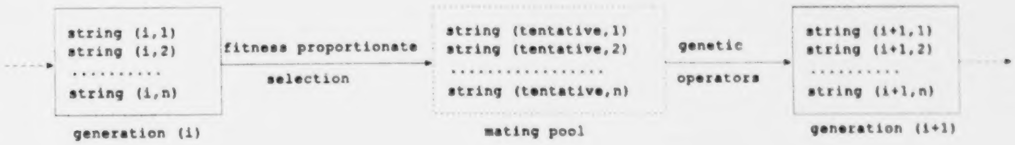

string (i,1)
string (i,2)
....
string (i,n)
generation (i)
fitness proportionate
selection
string (tentative,1)
string (tentative,2)
....
string (tentative,n)
mating pool
genetic
operators
string (i+1,1)
string (i+1,2)
....
string (i+1,n)
generation (i+1)

Figure 5-18: Evolution of populations in successive generations

The two commonly applied genetic operators are *crossover* and *mutation*. Crossover is the most important operator of a genetic-based technique. A simple one-point crossover scheme works as follows. Once a pair of strings is selected at random from the mating pool, an integer position *k* (called the *crossover site*) along the string is selected at random between 1 and *l*-1 where *l* is the length of the string in bits. Two new strings are now created by swapping all bits between position *k+1* and *l* inclusively. The "mating" process is repeated with other string pairs until the desired number of "child" strings are generated. In constant-population-size GAs, this number is the same as the original population size, in our example 5[9].

For instance, suppose that in the mating pool of the box design problem (Figure 5-17), strings 4 and 1 are randomly selected for mating. Each of the two strings has a length of 15 bits, so a crossover site must be picked berween 1 and 14. Let us suppose that bit 11 is randomly chosen and the strings are partially swapped from this point (Figure 5-19).

string 4 00100 01100 1 | 1000 ===) 00100 01100 10100

string 1 01010 11100 0 | 0100 01010 11100 01000

Figure 5-19: *Crossing over* of strings 1 and 4 in Figure 5-17

Crossover results in a randomized, yet structured information exchange. Each "child" string combines the characteristics of its "parent" strings. Considering the fact that in every search procedure there is a tradeoff between creating new knowledge and exploiting the already existing knowledge, one can regard crossover as the means for exploiting the existing knowledge in GAs. By combining chromosomes to form *string patterns* that may not have previously appeard in the population, crossover also provides a mechanism for exploring new regions of the search space.

---

[9] Since child-strings are generated in pairs, the process would not be able to generate an odd number of offspring. Therefore in the case of an odd population size one may generate one more (less) string and then eliminate (add) a single string to the population. One way to do this is to remove the least-fit chromosome from the population (add a duplicate of the most-fit chromosome to the population). In this introduction to GAs we shall not discuss this any further.

New knowledge can be introduced to the system by applying a second genetic operator called *mutation*. Mutation basically involves the random alteration of a bit (0 to 1 or 1 to 0) in a randomly chosen string. The operator is normally applied to post-crossover strings in the mating pool. Again, a *mutation site* is randomly selected along the string (between bits 1 and *l* inclusive) and the respective bit is switched. Mutation introduces a type of occasional random walk in the search space and prevents the system from being trapped in local optima. Mutation also allows for the formation of string patterns that may not have been present in the initial, randomly generated *finite* sized population.

Let us see how mutation affects our crossed-over strings of Figure 5-19. Suppose that a random integer generator gives us 4 as the site of mutation. In the first string on the right hand side of Figure 5-19, the 4th bit (from left) is a 0. Changing this to 1 will alter the string as shown in Figure 5-20.

(mutation site)
↓
00100 01100 10100 → 00110 01100 10100

Figure 5-20: Mutation of a gene

The results of genetic operations "crossover" and "mutation" on the two "parent" strings are summarized in Figure 5-21. Both parent- and child strings are shown along with their fitnesses. Note that one of the child strings happens to be invalid and is rejected according to the area-constraint. Nonetheless, the other one exibits a higher fitness than both its parents. This phenomenon is typical of GAs, as their power lies in improving the average fitness of successive generations.

Continuing with our box design example, in Figure 5-22 we show a typical generation 1 of strings, created through application of fitness-proportionate reproduction, crossover and mutation to the population of generation zero. The process has been repeated until a required number (5) of new strings have been generated. A comparison between the fitnesses of the two consecutive generations (Figures 5-15 and 5-22) shows that the average fitness of the generation has increased from 1192.2 to 1382.4.

Parent Strings	Fitness	Child Strings	Real Values	Fitness
00100 01100 11000	1152	00110 01100 10100	6" × 12" × 20"	1440
01010 11100 00100	1120	01010 11100 01000	10" × 28" × 8"	invalid

Figure 5-21: Results of a reproduction iteration

Proceeding to succeeding generations, the average fitness of the population will further improve and the global optimum solution (12" × 12" × 12") will be found (in this simple problem) in a few generations. One should note that our assertion about the growing average fitness of the consecutive generations does not imply an all-fit cast of strings in all generations. Many weak individuals may, and usually do, appear in one generation or another, but according to the principle of "survival of the fittest" they soon are replaced by stronger individuals and "die out". In GAs this "fate" materializes in the form of fitness-proportionate reproduction whereby the weaker chromosomes get little chance to be selected to have any offspring in the following generation.

The evolutionary process is terminated when convergence is detected or when another termination criterion (such as processing of a certain number of generations) is met (Figure 5-23). Convergence in the context of GAs is measured by the uniformity of the fitnesses of the strings in a population. A common termination criterion is that 95% of these strings share the same fitness, or the average fitness of the population falls within 95% of the maximum fitness in the same population (Dejong 1975).

String No.	String (Chromosome)	Real Values	Fitness
1	00110 01100 10100	6" × 12" × 20"	1440
2	11110 00111 00110	30" × 7" × 6"	1260
3	00100 11100 01010	4" × 28" × 10"	1120
4	01000 10111 01111	8" × 23" × 8"	1472
5	01001 01010 10010	9" × 10" × 18"	1620
Total			6912
Average			1382.4

Figure 5-22: Generation 1 (reproduced from generation zero, Figure 5-15)

Having introduced the basic idea of GAs, it is now time to take a closer look at some of the elements that give the trivial-looking method its power and robustness. We shall also explain briefly how the values of such GA parameters as string length, population size, crossover rate and mutation probability are determined.

The string length specifies the maximum number of discrete values a string can represent (the *string capacity*). When binary strings are used, this number is $2^l - 1$ where l is the length of the string in bits[10]. It must be clear by now that standard (as opposed to special purpose) GAs work only with discrete values of variables. Domains of continuous variables therefore need to be *discretized* before applying GAs to them. The finer the discretization of the domain, the longer the string required to represent it.

In cases where the formula ($2^l - 1$ = *number of discrete values of the variable*) does not yield an integer value for l, the next nearest integer value is given to l and the rest of the variable's domain is filled with repetitions of existing values. The same thing happens when various variables of a problem have different number of values in their domains and a unique string length is to be used to represent them all, for convenience purposes. In this case the string length is calculated based on the most crowded variable domain and the domains of the other variables are arbitrarily filled with other values in those domains[11].

Suppose, for example that a variable has 10 discrete values in its domain, say [1, 2, ..., 10]. We know that a string of length 3 can only represent 7 (= 2^3 - 1) values and a string of length 4 can represent 15 (= 2^4 - 1) values. We will therefore pick the latter and fill the domain randomly, say, as follows.

[1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 9, 9, 10, 10]

Also suppose that in a problem the domains of the three variables are as follows.

[20, 21, 22, 23, 24, 25, 26, 27, 28] (9 values)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] (15 values)

[30, 31, 32, 33, 34, 35, 36] (7 values)

Although the 7 values of the third variable can well be represented by a 3-bit string, in order to use a uniform string length, we would use strings of length 4 (required by the second

---

[10] Some researchers (including Goldberg 1989) prefer to use the full mapping capacity of a binary string (2^l rather than 2^l - 1), meaning that they include in their mapping the all-zero binary string (equivalent of decimal zero) too. We, however, have chosen to map only the non-zero codes (after Jenkins 1991) for programming convenience.

[11] From a probabilistic point of view, this arbitrary duplication of a subset of the discrete values in a variable's domain will not affect the results of the stochastic search, for all the values in the domain of the variable are given the same "duplication probability". Further elaboration of the issue is beyond the scope of this report.

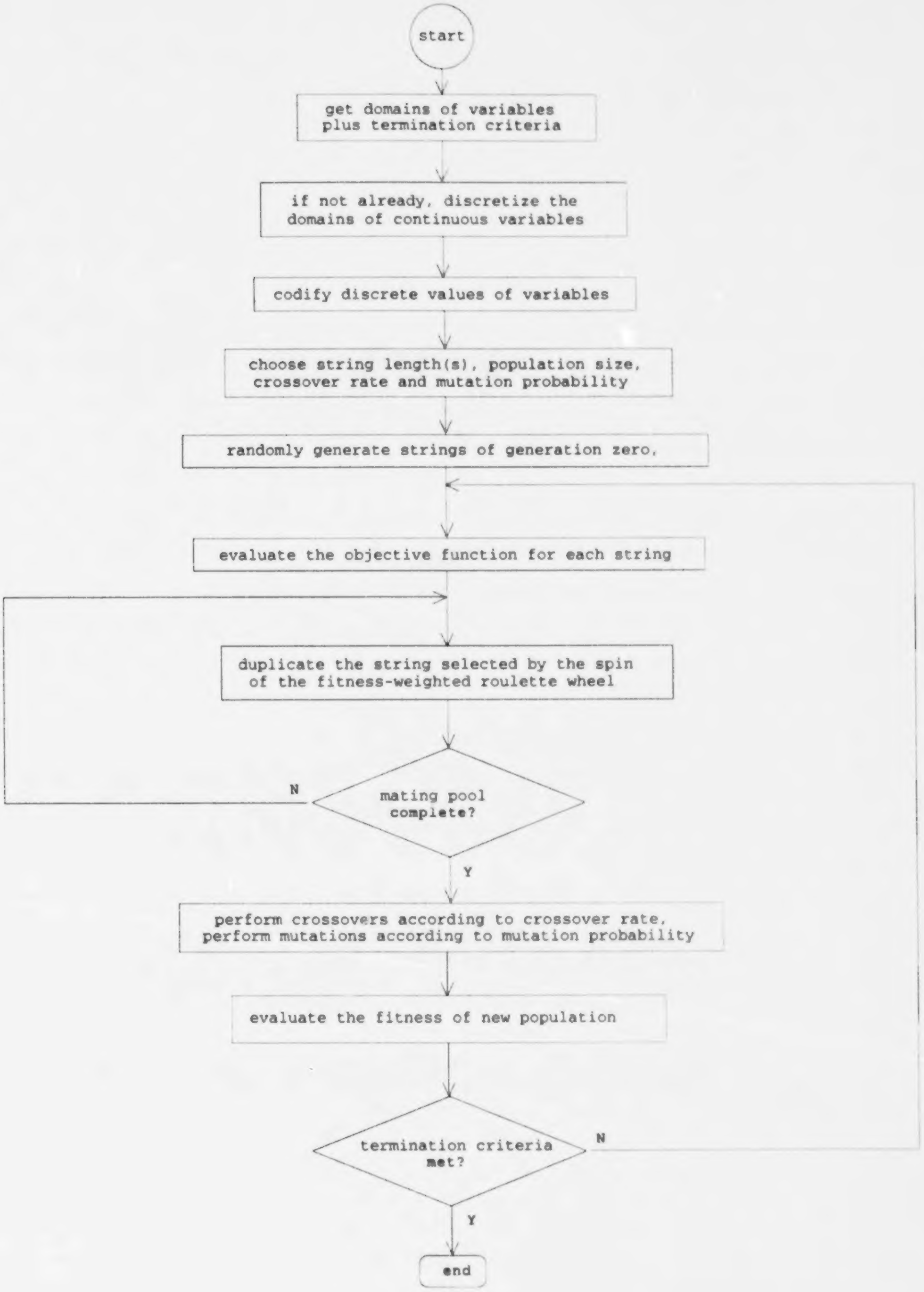

start
get domains of variables plus termination criteria
if not already, discretize the domains of continuous variables
codify discrete values of variables
choose string length(s), population size, crossover rate and mutation probability
randomly generate strings of generation zero,
evaluate the objective function for each string
duplicate the string selected by the spin of the fitness-weighted roulette wheel
mating pool complete?
N
Y
perform crossovers according to crossover rate, perform mutations according to mutation probability
evaluate the fitness of new population
termination criteria met?
N
Y
end

Figure 5-23: Basic operation of *Genetic Algorithms*

variable) to represent all three variables as follows.

[20, 20, 21, 21, 22, 22, 23, 24, 25, 26, 26, 27, 27, 28, 28] (15 values)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15] (15 values)
[30, 30, 31, 31, 32, 32, 33, 33, 33, 34, 34, 35, 35, 36, 36] (15 values)

Population size, or the number of strings in each generation, directly affects the ability of GAs to find the global optimum and their rate of convergence. A too small population will converge too quickly and often to suboptimal solutions. This is termed *Genetic Drift* and is, in the biological world, analogous to in-breeding within small, closed populations of organisms. The larger the population, the more points in the solution space being examined at each iteration and hence the higher the chance of hitting the global optimum in fewer iterations. However, a too large population results in long waiting times for any significant improvement and fewer chance of good strings to mate due to crowded populations.

Empirical studies have pointed toward the existence of an optimal population size for each string length. Goldberg (1985) suggests the following relation for the calculation of optimal population size as a function of string length (for binary-coded GAs with string lengths of up to 60).

$$popsize_{opt} = 1.65 \times 2^{0.21l}$$

where l is the string length. Based on this relation, he recommends the following population sizes for some of the more common string lengths.

String Length	Population Size	String Length	Population Size
4	4	18	22
5	4	19	26
6	5	20	29
7	5	21	34
8	6	22	38
9	7	24	51
10	8	25	58
11	9	26	67
12	10	28	77
14	13	30	88
15	15	35	101
16	17	40	116

Crossover and mutation play crucial roles in a GA seach, especially when they are applied at the same time. When only mutation is applied, the search resembles a random search. On the other hand, when only crossover is applied, the system may quickly spot a suboptimal solution (local optima) that exists within the initial population, thus resulting in a premature convergence. The tradeoff between the two operators allows GAs to successfully converge in most cases to the global optimum.

Operation of the two operators is controlled by their prescribed rates or probabilities (expressed as a percentage). These probabilities are either specified by the user or determined by the system according to some given criteria. In a programming sense, they tell the system at each point how many strings must be crossed over and how many must be mutated. Suppose for example that in our box design problem (population size = 5) we have a crossover probability of 80%. This means that 4 (= 0.8 × 5) of the strings have to be crossed over and a fifth one will directly go to the next generation.

Also, a mutation probability of 1% in the same problem means that on average, one bit will be switched somewhere in the population every other generation. This is because it will take a gene a minimum of 7 strings (more than one population size) before its probability of mutation becomes ≥ 1 ($15 \text{ bits/string} \times 7 \text{ strings} \times 0.01 = 1.05 \geq 1.0$).

Various researchers have recommended a value of 0.6 to 1.0 for the crossover probability and a value of less than 0.05 for the mutation probability (Grefenstette 1986). Due to the importance of these genetic parameters in a GA search and their direct influence on its rate of convergence, we would inevitably need to find optimal values of the parameters for our search purposes. In *Design by Exploration* this optimization *is* somewhat possible as we are dealing with a specific class of problems, namely instantiation of mechanical components, which maintains the same formulation for all the problems in the class.

5.2.5.2 *TUNING* THE GENETIC PARAMETERS FOR DbE

a. *Crossover Rate*

Three different example problems were chosen as a testbed for determining the optimal values of genetic parameters. The problems were to design a spur gear drive, a helical compression spring and a rotating shaft. All three problems were formulated as explained earlier in this section. The formulations of the problems are presented in Appendix E.

To solve the problems, a genetic algorithm was applied to them separately. Each problem was tried with three different sets of initial specifications and for each set the program was run 50 times (a total of 150 runs per problem). The test was repeated for values of Crossover Rate (CR) between 0.6 and 1.0 inclusive with a stepsize of 0.1 and for values of Mutation Probability (MP) between 0.005 and 0.05 in various stepsizes (0.005, 0.01, 0.02, 0.03, 0.04, 0.05). In each case the population size was selected as per Goldberg's recommendations (Goldberg 1985) presented in the previous section.

For each value of CR, two indices were calculated and recorded:

- The number of successful convergences (i.e. to a zero penalty function) out of 150 runs (per MP value), averaged over the range of MP values, and
- The average number of generations to converge in cases of successful convergence.

In order to be comparable, the results from different trials were normalized and transformed into a *Measure of Merit* defined as

$$M_M^i = N_C^i \times \frac{1}{\dfrac{\sum_{j=1}^{N_C^i} N_G^j}{N_C^i} \,.\, P^i} = \frac{(N_C^i)^2}{\sum_{j=1}^{N_C^i} N_G^j \,.\, P^i} \qquad (5\text{-}3)$$

where:

M_M^i = *average measure of merit of the* i^{th} *problem*

N_C^i = *average number of successful convergences out of* 150 *runs*

P^i = *population size of the* i^{th} *problem*

N_G^j = *average number of generations to converge in the* j^{th} *problem*

This measure of merit basically represents the rate of successful convergence per processed design, as the first denominator in Equation 5-3 is simply the "average number of generations to converge" times the "number of designs to be processed in each generation". M_M is plotted against CR values for the three problems in Figure 5-24. The average values of CRs for the three problems are also shown. As the plotted values show, the crossover rate of 1.0 can be considered the optimal value for the tested problems. We shall therefore use this value in our implementation of GAs.

b. *Mutation Probability*

Genetic algorithms are always at risk of premature convergence to suboptimal points in the solution space. The element of mutation is incorporated in GAs to prevent this from happening or at least to lower the risk of it. A too low mutation probability would normally mean a quicker convergence, but possibly to a suboptimal point. This is because with a low

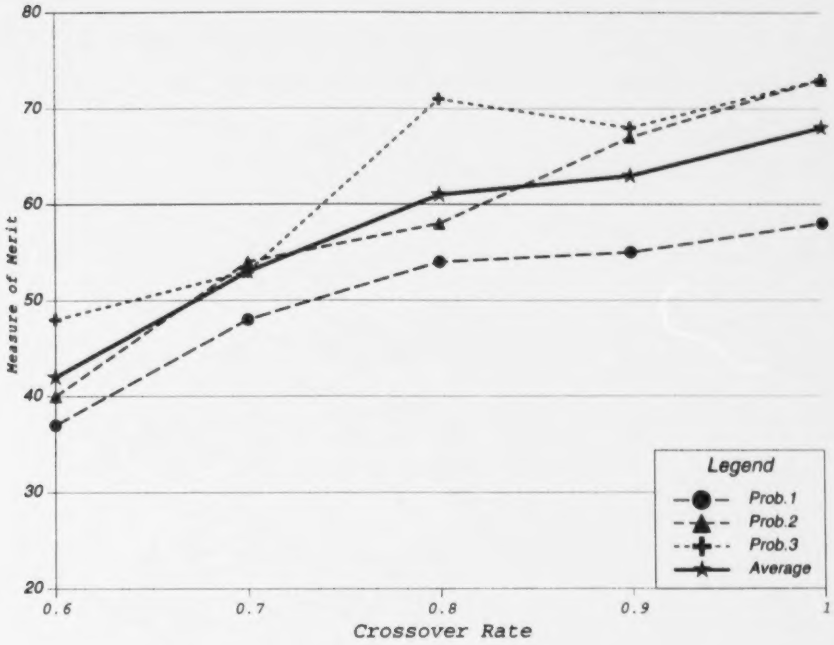


Figure 5-24: Results of tests to find optimal value of CR

MP, the system would not be forced to explore other areas of the solution space than the one it is currently in. A too high MP, on the other hand, would mean a higher chance to converge to global optimum but at the cost of having to process too many generations to improve the fitnesses and to discard the emerging, suboptimal points.

Furthermore we should note that a constant MP, no matter how carefully it might have been selected, would normally not maintain the same level of "optimality" and "desirability" during the entire course of a GA search. An MP value which may work perfectly for the first few generations of a search, may well turn into a nuisance in final generations. This is because at the beginning of a search we need more mutations to happen so that the system can explore larger areas of the solution space and locate the more promising regions, whereas at final steps we need the generations to be left undisturbed and perturbation-free so that they can smoothly approach the optimum. In other words, it would be ideal if the mutation probability could change inversely with the change of generations' average fitnesses.

The above argument inspires the notion of a *fitness-dependent* mutation probability, that is, the MP decreasing with the increase of the generation's average fitness and vice-versa.

To keep things simple, in this work we assume a fitness-dependent mutation probability with an inverse linear relation between the value of MP and the average fitness of the generation. This relation is expressed as equation 5-4 and illustrated in Figure 5-25.

$$MP^{i} = MP^{0} \times \frac{1 - \dfrac{f^{i}_{av.}}{f^{i}_{max..}}}{1 - \dfrac{f^{0}_{av.}}{f^{0}_{max..}}} \tag{5-4}$$

where:

MP^{i} = ***mutation probability for generation i***
MP^{0} = ***initial mutation probability (generation zero)***
$f^{i}_{av.}$ = ***average fitness of generation i*** $\left(= \dfrac{\sum_{j=1}^{popsize} f^{i}_{j}}{popsize}\right)$
$f^{i}_{max..}$ = ***maximum itness of generation i***
$f^{0}_{av.}$ = ***average fitness of generation zero***
$f^{0}_{max..}$ = ***maximum fitness of generation zero***

The results of the tests reported in subsection (a) above indicated that an initial MP value of 0.02 would give the highest average range of convergence for the problems considered. Therefore equation 5-4 was modified as follows (Equation 5-

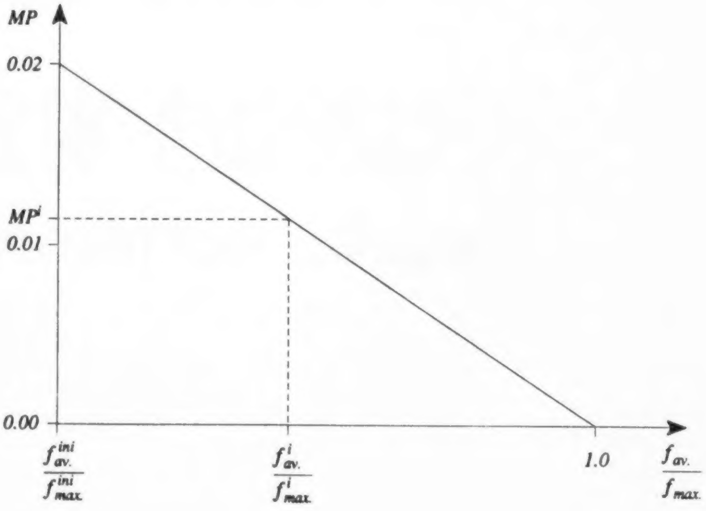


Figure 5-25: Generation-dependent MP

5).

$$MP^{i} = 0.02 \times \frac{1 - \dfrac{f^{i}_{av.}}{f^{i}_{max..}}}{1 - \dfrac{f^{0}_{av.}}{f^{0}_{max..}}} \qquad (5\text{-}5)$$

To complete our discussion of the GA parameters used in our applications, we should mention another point or two. First, that a *stochastic remainder selection scheme without replacement* was found to result in the highest performance of the GA and was used in the development of the exploration module. A *selection scheme* tells us how "fitnesses" of individual strings in a generation are translated into the number of copies they are going to have in the mating pool. Unlike the standard *weighted roulette wheel* scheme explained earlier, here each string of the previous generation gets as many copies in the mating pool as the integer part of its "relative fitness" for sure, where the relative fitness of a string in a population is defined as (string fitness / average fitness of the population). Then, to fill the mating pool, the fractional part of the string's relative fitness is used to make a "biased coin" which is then tossed to determine whether another copy of this string will go to the mating pool. The coin-tossing process will continue until the pool is full.

Second, that to improve the convergence properties of the GAs employed in this work, a linear scaling of the raw fitnesses of the strings was used. Both of these decisions are in agreement with the recommendations of Goldberg (1989). A detailed description of the terms presented in this and the previous paragraphs can be found in the same reference.

5.2.5.3 CONVERGENCE ISSUES

The stochastic nature of GAs make theoretical assertions about their convergence properties very difficult[12]. Theoretically, there are no "sufficient conditions" for the convergence of GAs, neither is there a way to exactly predict when a GA with a given set of control parameters will, if at all, converge. However, there are proven theorems that explain the behavior of GAs and predict some aspects of it. A complete discussion of these theorems is beyond the scope of this report. Nonetheless for the benefit of the non-expert reader, here we outline the so-called *fundamental theorem of genetic algorithms* (Holland 1975).

The theorem, also called the *schemata theorem,* explains how GAs work and where their power comes from. To understand the theorem we need first to define a few terms.

---

[12] To our knowledge, no formal theoretical assertions have been made, to date, about these properties. Nonetheless, a number of sufficient, and not necessary, conditions for the convergence of GAs have been reported in (Bethke 1975).

- A *schema* (plural *schemata*) is a string-template expressed in the alphabet {0, 1, *} (in a binary coding), where a (*) represents a "wildcard" matching either a 0 or a 1. For instance, the schema (10***) represents all the strings beginning with 10 with a length of five. Similarly, the schema (11*10) represents the two strings (11110) and (11010).
- The *order* of schema H, denoted by "O(H)", is the number of its specified (non *) bits. The order of schema (10***), for example, is two and that of (11*10) is four.
- The *defining length* of schema H, denoted by δ(H), is the distance between its outermost specified bits. The defining length of the schema (10***) is one (= last specified bit, 2, minus first specified bit, 1) and that of (11*10) is four.

Using the above definitions, the fundamental theorem of GAs can now be expressed as follows.

$$n(H,\ i+1) \geq n(H,\ i) \times \frac{f^{i}_{av.}(H)}{f^{i}_{av.}} \times \left[\ 1 - CR^{i} \times \frac{\delta(H)}{l-1} - O(H) \times MP^{i}\ \right] \qquad (5\text{-}6)$$

where

$n(H,\ i+1)$ = ***number of strings matching schema H in generation (i+1)***
$f^{i}_{av.}(H)$ = ***average fitness of strings matching schema H in generation (i)***
$f^{i}_{av.}$ = ***average fitness of generation (i)***
CR^{i} = ***rate (probability) of crossover in generation (i)***
l = ***length of schema (H)***
MP^{i} = ***probability of mutation in generation (i)***

Asserting that the fitness of a string depends on the goodness of the schemata it contains, the theorem explains the secret of GAs' success as follows. *Stronger schemata (the ones with above-average fitnesses) receive exponentially increasing instances in successive populations. This means that fitter individuals have more chance to have offspring in succeeding generations.*

Also, the theorem states that a particular (good) schema will have a better chance of survival if it has a low order and short defining length, as the former means less chance of losing one's good genes to mutation and the latter means less chance of being disrupted by crossover. The derivation of the above theorem can be found in (Goldberg 1989).

This is pretty much as far as we can go on the theoretical front. Despite the lack of a solid theoretical proof for the convergence of GAs, results of empirical studies including an abundance of successful applications (including our own) have established GAs as robust general purpose search techniques (see Beasley, Bull and Martin 1993 for a list of related literature.) Moreover, we have conducted our own investigation into how to promote the robustness and the convergence properties of the GAs used in this work by *tuning* their control parameters.

One of the results of this investigation was that no matter how precisely the GA parameters were selected, the possibility of the system converging to suboptimal solutions could not be completely eliminated. For example, in the tests described earlier in this section, an average 11% of the runs converged to suboptimal points.

To improve the rate of optimum-convergence (i.e. to the optimal solution), we devised a *stimulating* mechanism for the ill-converged cases whereby we "reboot" the search through "agitating" the final population and scattering its strings across the solution space. This basically is a shortcut to re-starting the search with new initial populations in the hope that the next convergence will be to the right point(s). To do this, we assign a sufficiently high value to the mutation probability of the dead-locked population and continue the course of reproduction. If the MP value is selected correctly, the next population will typically comprise points from all across the solution space and the search could be started afresh[13].

Our experience with the three test problems showed that a value of 4 to 6 times the original initial MP value (0.02) would give satisfactory results. Not to mention that higher values of MP would be preferable, but the choice of higher values would not be justifiable considering the time required to perform the mutation operations. A value of 0.1 (5×0.02) was therefore selected as the "reboot" value of mutation probability. The stimulating mechanism is graphically presented in Figure 5-26. The diagram of Figure 5-26 is a generalization of the one in Figure 5-25 to accommodate cases of suboptimal-convergence as well.

A question arises here as how the "user-independent" *conceptual designer* will decide whether or not the system has converged to the optimal solution, as this would require the

---

[13] We have chosen this stimulating mechanism over simply starting the search over to take advantage of the fact that in the reported experiment, the runs very often converged to points in a close neighborhood of the global optima. Considering that the stimulation does not usually affect all the chromosomes in the dead-locked population, the next population will most likely contain some of the points neighboring the global optimum. This in turn guarantees that the target region (the region encompassing the global optimum) will be explored. It also saves the GA a number of generations to generate a point in the target region (if at all), as a random new population (as prescribed by the start-over scenario) will not necessarily contain a point in that region.

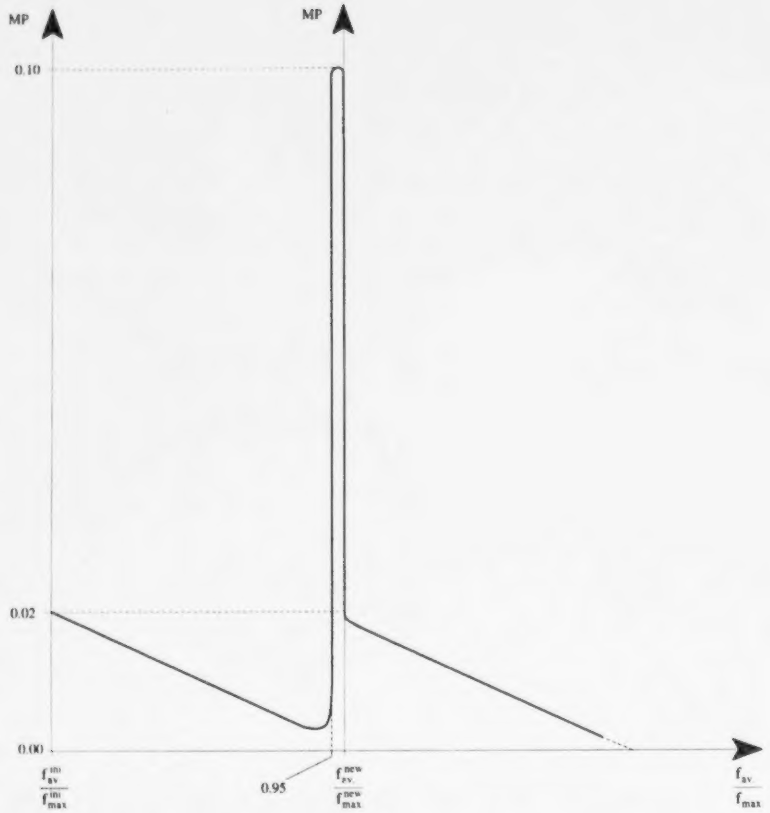


Figure 5-26: Rebooting mechanism for a dead-locked search

system to know the optimal solution *apriori.* The answer is that as we explained in our formulation of the "exploration" problem, in all cases we do know the optimum solution (zero for the value of the penalty function) already. As a matter of fact, this is one of the reasons for us calling the problem a pseudo-optimization one. Hence all the system has to do is to check whether the search has actually converged to zero or not.

The next question is "what if the stimulation doesn't work?", i.e. what if after rebooting the search, the system still converges to a suboptimal solution? An obvious answer would be to repeat the stimulation until we get the correct solution. This, however, might cause the system to get trapped in an endless loop as there might not exist a solution at all, in which case the system would reboot over and over without any improvement.

To avoid this, we define a stimulation-termination criterion and make the system quit and report failure once the termination condition is met. To develop the criterion we argue

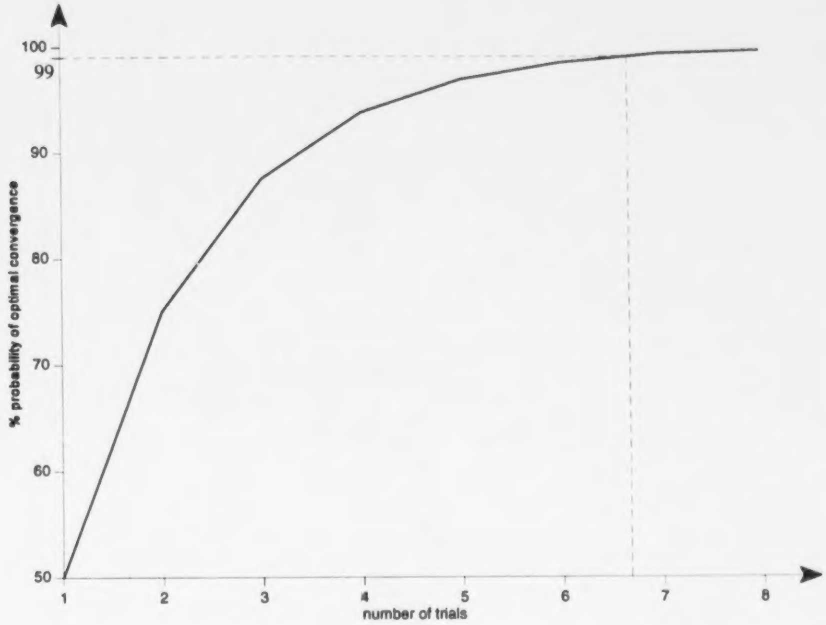


Figure 5-27: Probability of optimal convergence vs. number of trials

that a typical GA search has, at the worst, a 50% chance of a suboptimal convergence[14]. This probability will reduce to 25% (or a 75% chance of an optimal convergence) after two runs (the initial run and one rebooted run). The probability of optimal convergence can be calculated in succeeding trials using the following formula.

$$P^{i}_{conv.} = 1 - (\frac{1}{2})^{i} \qquad (5\text{-}7)$$

where $(P^{i}_{conv.})$ is the probability of convergence to the optimal solution in *i*th trial.

Equation 5-7 is graphically illustrated in Figure 5-27. Considering a 99% probability of optimal convergence $(P^{i}_{conv.} \geq 0.99)$ "safe" enough, equation 5-5 gives us a value of 7 for the number of trials, that is, it will take 7 trials for the probability to exceed 99%. Using this criterion, we limit the number of search-reboots to 6 (= seven minus the initial run).

---

[14] In practice, the probability of such ill-convergence is never this high. As mentioned earlier, our own experience with sample design problems resulted in an average 11% suboptimal convergences. Also, to our knowledge no one has reported this to occur at a rate as high as 50%.

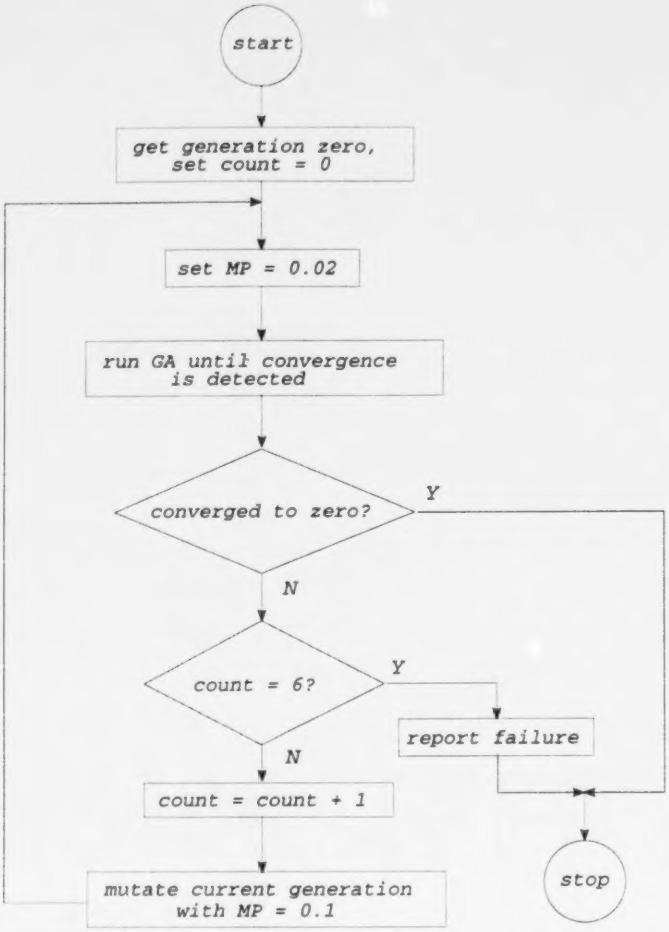

start
get generation zero, set count = 0
set MP = 0.02
run GA until convergence is detected
converged to zero?
Y
N
count = 6?
Y
N
report failure
count = count + 1
mutate current generation with MP = 0.1
stop

Figure 5-28: Modified GA with rebooting mechanism

Briefly then, the GA search of the exploration module works as follows (Figure 5-28).

1- Start with an initial mutation probability of 0.02 and a generation-dependent MP,
2- Proceed until convergence is achieved (the case of non-convergence has already ben discussed),
3- Check the solution (corresponding to the maximum fitness in the last generation) with the expected one (in this case zero), if they match then stop otherwise
4- Mutate the strings of the current generation with a probability of 0.1 (reboot the search),

5- Set the MP of the new generation back to 0.02 and go to step 2 above,

6- Repeat theprocess up to 6 times, if the search still fails then quit and report failure.

Application of this rebooting mechanism to our test problems resulted in the reduction of their average suboptimal convergences from 11% (of all convergences) to 2%.

To wrap up our discussion of GAs, we should point out a few observations here.

1- GAs show their power and dominance over other general-purpose search techniques in problems with large solution spaces. For small to medium size problems there are often other, more efficient techniques.

2- Regardless of the size of the problem, there might be specialized techniques for particular problems that out-perform GAs in both speed and accuracy (Figure 5-29). The main ground for GAs is then those difficult areas in which no specialized techniques exist. To this we should add those cases where the characteristics of the problem are not exactly known *apriori* and a robust, universal solver is needed (e.g. in *conceptual designer*).

3- Standard GAs work with discrete variables only. This being the case, the only way to avoid "missing" some solutions due to their being located "between" discrete values is to refine the discretization mesh accordingly. The finer the mesh, the longer the chromosome lengths and family sizes, and hence usually the more accurate the GA search, though this comes at the cost of more computations.

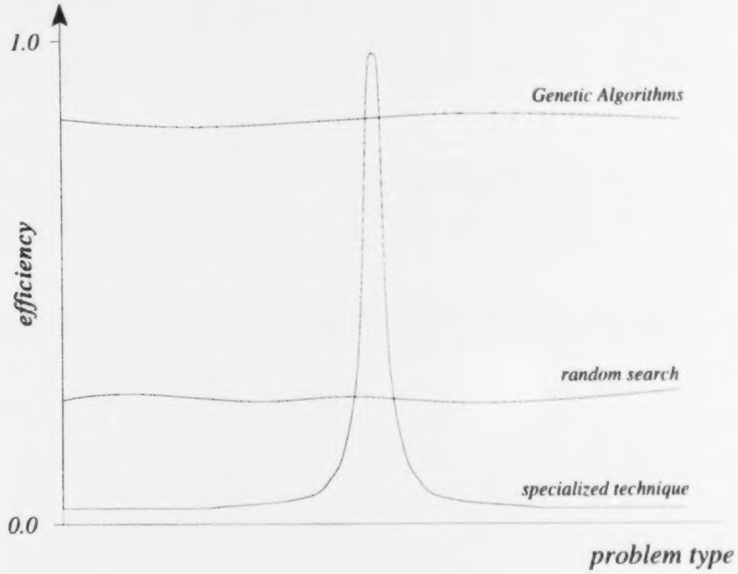


Figure 5-29: Efficiency of GAs compared to other search techniques (after Goldberg 1989)

4- GAs work on a *population* of possible solutions at the same time. This, plus the application of mutation, helps the technique find *multiple* feasible solutions (if they exist) across the solution space.

5- GAs are stochastic methods. Unlike deterministic methods which, given the same initial values, will find the same solution, GAs may converge to a different (or at least partially different) set of solutions under the same conditions.

5.2.5.4 PARTITIONING THE EQUALITY SUBSET

Having completed the presentation of genetic algorithms, we shall now resume our discussion of the problem-solving strategy of the *exploration module*. The strategy, so far, tells us to

- examine the constraint set of the problem for consistency and solvability,
- reformulate it as a pseudo-optimization problem,
- find its degrees of freedom and specify the "free variables",
- search the space of these free variables using a GA in order to find feasible values of these variables. "Feasible" values are the ones that when substituted in the equality constraints and solved for other (assigned) variables, collectively minimize (zero) the penalty function constructed from inequality constraints.
- once the problem is solved, report the solution(s) back to the functional representation panel of the blackboard.

As the above steps indicate, each iteration of the process involves the solution of the component's equality constraint-subset to find the values of the assigned variables. Very often, the equality constraints have a sparse incidence matrix, that is, there are many equations and each equation contains only a few variables. This is natural of mechanical designs, as each of their equations is usually intended to describe a single physical/functional aspect of them, among many. Computation time required to solve such an equation set can be significantly reduced, especially in case of large sets, by breaking the set into smaller, irreducible subsets of equations to be solved simultaneously. Because of this, we add one more step (called partitioning) to the previous steps of the exploration process with the intention to speed it up as much as possible.

Remember the reduced, square incidence matrix of section 5.2.2 representing an all-assigned set of m equations in m unknowns (one is shown in Figure 5-11 for the gear design example). Let us call this matrix SIM (for Square Incidence Matrix) for brevity. At this stage we plan to *partition* the SIM into a *block-triangular* matrix which will tell us the most efficient order of solving subsets of the equation set. A block-triangular matrix is one with square sub-matrices on its main diagonal and zeros above the diagonal otherwise. The sub-matrices on the diagonal will represent irreducible subsets of equations to be solved simultaneously.

The partitioning proceeds as follows (Figure 5-30).

1- Consider the SIM of the equation set. Generate a blank matrix of the same size and call it NEWSIM.
2- Starting with $I = 1$ and going up in steps of 1 ($I \leq$ order of SIM), look for I equations in J unknowns $(J \leq I)$.
3- Once such equations are found, remove them and the corresponding variables from SIM and enter them in NEWSIM in the order they are found.
4- Permute columns so that the assigned variable of each equation is located on the main diagonal.
5- Once completed, NEWSIM will represent the desired block-triangular version of the SIM.

Browsing through the flow diagram of Figure 5-30, it might seem that the algorithm could result in an overconstrained subset of equations, i.e. one with more equations than unknowns. This, however, will not happen; firstly because such an equation subset would presumably not make it through the "assignment" process (Figure 5-6) and into partitioning stage, and secondly because partitioning starts with $I=1$ and proceeds upwards, meaning that any subset of p equations in p unknowns would get probed before proceeding to one with $p+1$ equations in p unknowns.

Let us illustrate this by "partitioning" a hypothetical set of 10 equations in 10 unknowns represented by its incidence matrix in Figure 5-31. Like before, assignments (output variables) are specified by parentheses. As the matrix shows, on the average fewer than 3 variables (out of 10) appear in each equation, implying a highly sparse matrix.

First we generate a 10 by 10 blank (null) matrix (NEWSIM). Then we start the actual partitioning by looking for single equations in one unknown. Two such equations are found (equations 1 and 10). Equation 1 is now entered in the first row of NEWSIM and column 9 of SIM is entered as column 1 of NEWSIM so that x9 (output variable of equation 1) is now located on the main diagonal. Row 1 and column 9 are then removed from SIM. Similarly, equation 10 is entered in the second row of NEWSIM and its assigned variable, x10, is moved to column 2 and again the respective row and column are removed from SIM. The result is shown in Figure 5-32.

Looking at the reduced SIM (LHS of Figure 5-32), we now find another single equation (eq. 7) in one unknown (x1). Taking equation 7 to NEWSIM, bringing its output variable (x1) to the third column and removing row 7 and column 1 from SIM we will be left with a 7 by 7 SIM. Since there are no more single equations with one unknown in this new

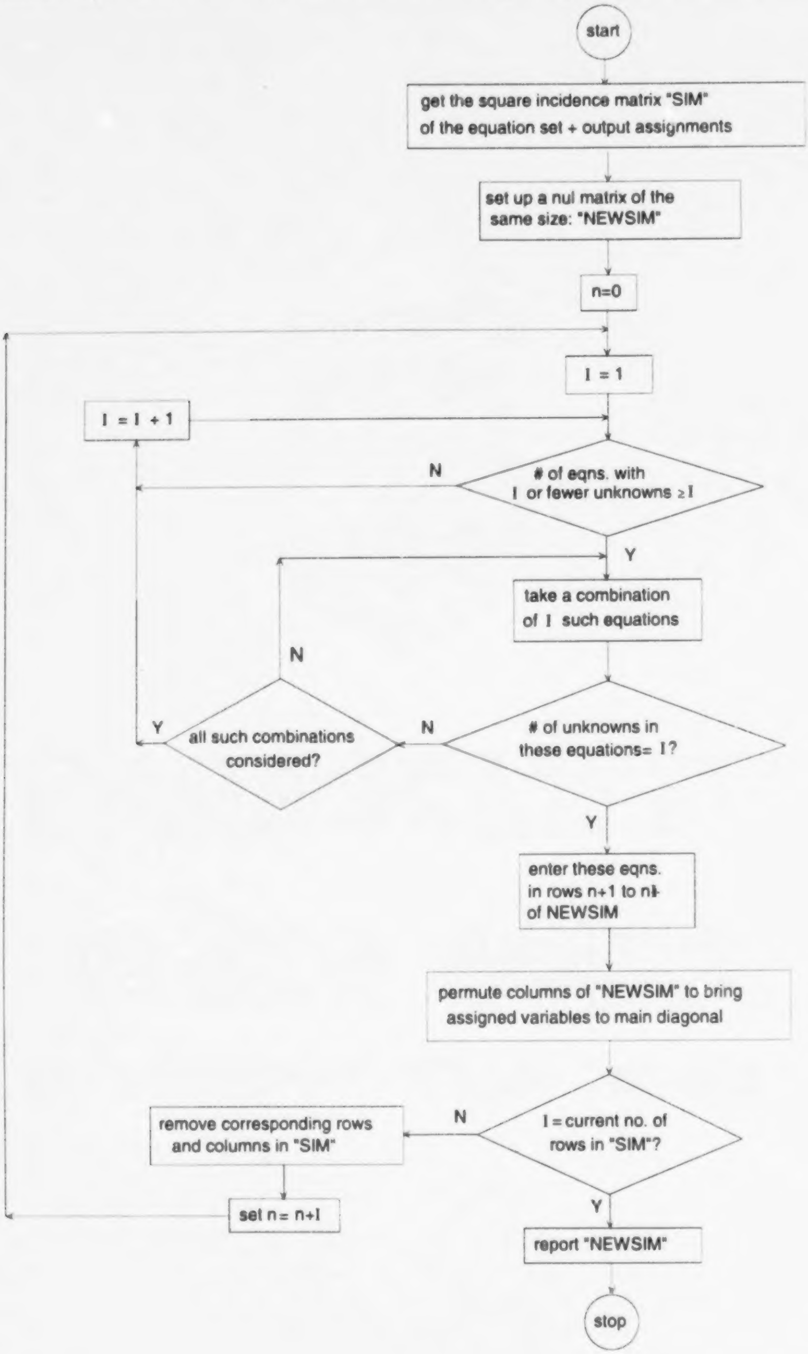

start
get the square incidence matrix "SIM"
of the equation set + output assignments
set up a nul matrix of the
same size: "NEWSIM"
n=0
I = 1
I = I + 1
N
# of eqns. with
I or fewer unknowns ≥ I
Y
take a combination
of I such equations
N
Y
all such combinations
considered?
N
# of unknowns in
these equations= I?
Y
enter these eqns.
in rows n+1 to n+I
of NEWSIM
permute columns of "NEWSIM" to bring
assigned variables to main diagonal
N
remove corresponding rows
and columns in "SIM"
I = current no. of
rows in "SIM"?
set n = n+I
Y
report "NEWSIM"
stop

Figure 5-30: Partitioning a sparse matrix

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
eq. 1	0	0	0	0	0	0	0	0	(1)	0
eq. 2	0	0	1	1	0	0	(1)	1	0	0
eq. 3	0	0	(1)	1	1	0	0	0	0	0
eq. 4	0	(1)	0	0	0	1	0	0	1	0
eq. 5	0	1	1	0	0	(1)	1	0	0	0
eq. 6	1	0	0	(1)	1	0	0	0	0	1
eq. 7	(1)	0	0	0	0	0	0	0	1	1
eq. 8	0	0	1	1	(1)	0	0	0	0	0
eq. 9	0	0	0	0	0	0	1	(1)	0	0
eq. 10	0	0	0	0	0	0	0	0	0	(1)

Figure 5-31: Incidence matrix (SIM) of an equation set

SIM, we now start looking for possible sets of two equations in two unknowns.

Equations 4, 6 and 9 each has 2 unknowns, but no two of them have the same variables as unknowns, that is, we cannot find a subset of two equations with the same two unknowns. Therefore we proceed to three and try to find 3 equations in 3 unknowns. Equations 3, 6 and 8 make such a set, with x3, x4 and x5 as their unknowns. These three equations are taken to rows 4, 5 and 6 of NEWSIM respectively and their assigned variables (x3, x4 and x5) are moved to columns 4, 5 and 6 of NEWSIM respectively. The three equations and their assigned variables are then removed from SIM (Figure 5-33).

The remaining SIM is a 4 by 4 matrix in which no single equation with one unknown exists. Three equations each with two unknowns can now be recognized (equations 2, 4 and 9), but only two of them (eq. 2 and eq. 9) are in the same two unknowns (x7 and x8). moving these two equations to NEWSIM we will be left with another two equations (eq. 4 and eq. 5), now in two unknowns (x2 and x6). Removing these to NEWSIM will complete the partitioning.

The resulting block-triangular incidence matrix is shown in Figure 5-34. One could see that other than the square blocks on the diagonal (distinguished by double lines), the upper triangle is basically zero. The matrix now tells us that the most efficient order to solve the set of equations represented by the matrix is as follows.

- Solve equation 1 for x9 and substitute the value of x9 in equations 4 and 7,
- Solve equation 10 for x10 and substitute the value of x10 in equations 6 and 7,
- Equation 7 has now only one unknown left (x1); solve it for its unknown and substitute

the value of x1 in equation 6,

- Solve equations 3, 6 and 8 for x3, x4 and x5 and substitute their values in any other equations they appear in,
- Solve equations 2 and 9 for x7 and x8 and substitute the value of x7 in equation 5, and
- Solve equations 4 and 5 for x2 and x6.

SIM											NEWSIM										
x → eq. ↓	1	2	3	4	5	6	7	8	9	10	x → eq. ↓	9	10	3	4	5	6	7	8	1	2
1											1	1	0	0	0	0	0	0	0	0	0
2	0	0	1	1	0	0	1	1			10	0	1	0	0	0	0	0	0	0	0
3	0	0	1	1	1	0	0	0													
4	0	1	0	0	0	1	0	0													
5	0	1	1	0	0	1	1	0													
6	1	0	0	1	1	0	0	0													
7	1	0	0	0	0	0	0	0													
8	0	0	1	1	1	0	0	0													
9	0	0	0	0	0	0	1	1													
10																					

Figure 5-32: First two steps in transforming SIM to NEWSIM

The triangularization has reduced the maximum size of the equation set(s) to be solved from 10 to 3, meaning a noticeable reduction in the amount of calculation required.

5.2.5.5 THE SOLUTION SCHEME AT A GLANCE

Because of the many diverse techniques and discussions presented in this section, here we feel we need to summarize the solution scheme employed by the exploration module for the benefit of the reader. The scheme can be presented in the following 7 steps (Figure 5-35).

1- Having received the requirements and specifications of the problem, form the knowledge pool of the component (section 5.2.1).

S I M											N E W S I M										
x → eq.↓	1	2	3	4	5	6	7	8	9	10	x → eq.↓	9	10	1	3	4	5	7	8	6	2
1											1	1	0	0	0	0	0	0	0	0	0
2		0				0	1	1			10	0	1	0	0	0	0	0	0	0	0
3											7	1	1	1	0	0	0	0	0	0	0
4		1				1	0	0			3	0	0	0	1	1	1	0	0	0	0
5		1				1	1	0			6	0	1	1	0	1	1	0	0	0	0
6											8	0	0	0	1	1	1	0	0	0	0
7																					
8																					
9		0				0	1	1													
10																					

Figure 5-33: Incidence matrices after four transactions

2- Extract the overall constraint set (C) of the problem and separate the equality (H) and inequality (G) subsets (section 5.2.2).

3- To make sure the problem is well constrained and hence potentially solvable, examine the equality subset: find its maximal matching and determine the free variables whose specification would render the set exactly constrained. If not possible to do so, report failure and quit (section 5.2.2).

4- To facilitate the frequent solution of the equality set, "partition" the set and determine the order in which smaller, irreducible subsets of equations can be solved (section 5.2.5.4).

5- Reformulate the problem as a pseudo-optimization one: form a penalty function (from the inequality constraints) subject to equality constraints, to be zeroed by tuning the free variables (Section 5.2.4).

6- Discretize the domains of free variables if not already, and conduct a genetic search in their hyper-space to find set(s) of feasible values that would satisfy the equality set and zero the penalty function. Each of these sets would represent a feasible instance of the component at hand (section 5.2.1).

7- Report specifications of the feasible instances back to the blackboard. If none has been found, report failure and quit.

x → eq. ↓	9	10	1	3	4	5	7	8	2	6
1	(1)	0	0	0	0	0	0	0	0	0
10	0	(1)	0	0	0	0	0	0	0	0
7	1	1	(1)	0	0	0	0	0	0	0
3	0	0	0	(1)	1	1	0	0	0	0
6	0	1	1	0	(1)	1	0	0	0	0
8	0	0	0	1	1	(1)	0	0	0	0
2	0	0	0	1	1	0	(1)	1	0	0
9	0	0	0	0	0	0	1	(1)	0	0
4	1	0	0	0	0	0	0	0	(1)	1
5	0	0	0	1	0	0	1	0	1	(1)

Figure 5-34: The block-triangularized incidence matrix NEWSIM

As promised earlier, we shall now resume our examination of the gear-design example which we left off in Section 5.2.2. By then we had gone as far as determining the problem's free variables and making sure the problem is (potentially) solvable. The free variables were determined to be the face width of the gear (FW), the gear module (ML), the number of pinion teeth (NPT) and the allowable stress of the gear material (AlS).

As prescribed by item 4 above, the next step is to partition the square incidence matrix of the equality set (Figure 5-11). The outcome is the block-triangular incidence matrix shown in Figure 5-36. Note that in this particular case the resulting matrix is simply lower-triangular, meaning that once values of the free variables are specified and substituted in the equation set, the equations may be solved one by one. As a matter of fact, this particular arrangement tells us that as long as equations 3, 7 and 8 are solved before equation 6, equations can be solved in any arbitrary order and the ordered incidence matrix is not unique.

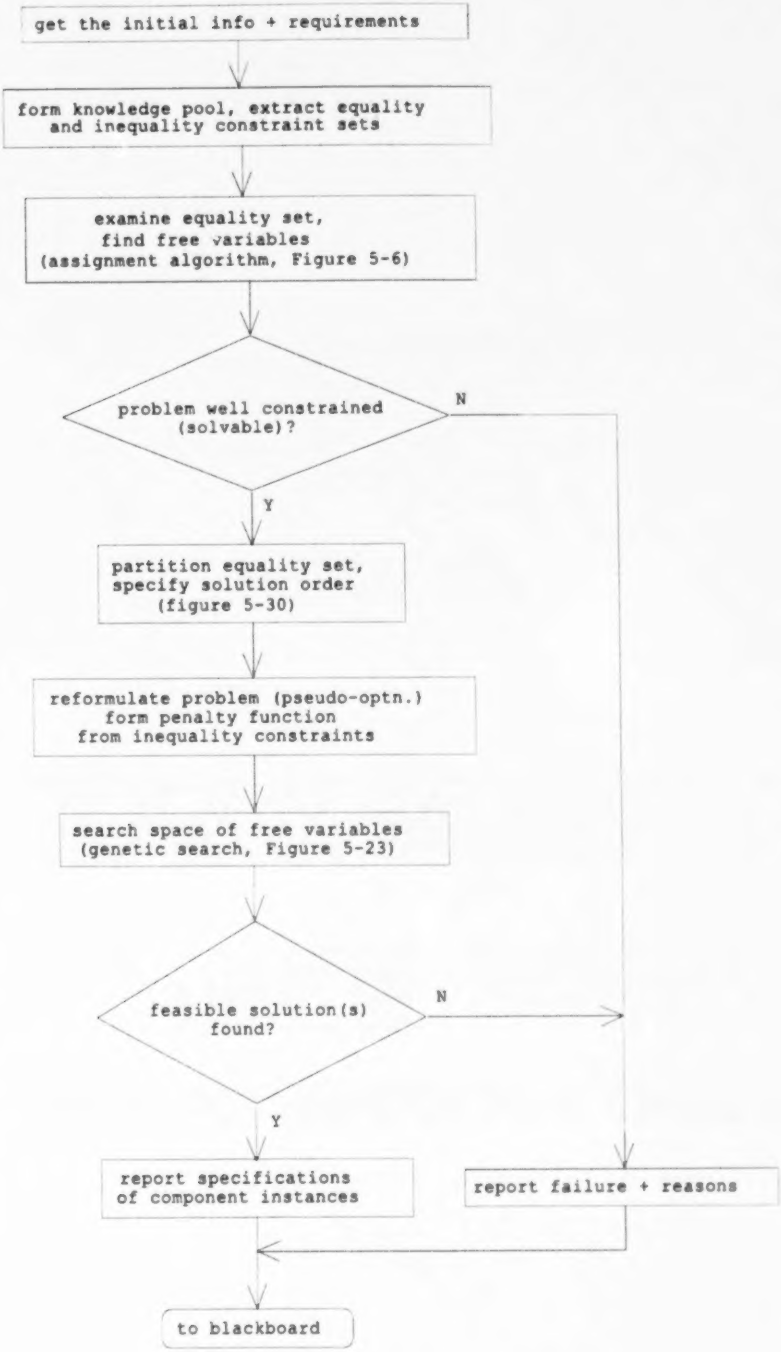

get the initial info + requirements
form knowledge pool, extract equality and inequality constraint sets
examine equality set, find free variables (assignment algorithm, Figure 5-6)
problem well constrained (solvable)?
N
Y
partition equality set, specify solution order (figure 5-30)
reformulate problem (pseudo-optn.) form penalty function from inequality constraints
search space of free variables (genetic search, Figure 5-23)
feasible solution(s) found?
N
Y
report specifications of component instances
report failure + reasons
to blackboard

Figure 5-35: Operation of *Exploration Module*

$$
\begin{array}{lcccccccc}
 & \mathrm{AT} & \mathrm{CD} & \mathrm{TL} & \mathrm{OS} & \mathrm{NGT} & \mathrm{K_v} & \mathrm{Y} & \mathrm{MBS} \\
\text{eq.1} & (x) & . & . & . & . & . & . & . \\
\text{eq.2} & . & (x) & . & . & . & . & . & . \\
\text{eq.3} & . & . & (x) & . & . & . & . & . \\
\text{eq.4} & . & . & . & (x) & . & . & . & . \\
\text{eq.5} & . & . & . & . & (x) & . & . & . \\
\text{eq.7} & . & . & . & . & . & (x) & . & . \\
\text{eq.8} & . & . & . & . & . & . & (x) & . \\
\text{eq.6} & . & . & x & . & . & x & x & (x)
\end{array}
$$

Figure 5-36: The ordered (block triangular) incidence matrix of the spur gear drive

Proceeding to item 5 above and trying to reformulate the problem, we now consider the inequality constraints of the problem (Figure 5-12 and repeated here as Figure 5-37 for convenience).

$$CD - 0.4m \leq 0$$

$$0.2m - CD \leq 0$$

$$SR - 8.0 \leq 0 \quad (SR > 0)$$

$$FW - 16\ ML \leq 0$$

$$9\ ML - FW \leq 0$$

$$MBS - AlS \leq 0$$

Figure 5-37: Gear drive's inequality constraint subset (**G**)

Using equation 5-2, we transform these constraints into the following penalty function (equation 5-8).

$$P.F. = [\max(0, \frac{CD-0.4}{0.3})]^2 + [\max(0, \frac{0.2-CD}{0.3})]^2$$
$$+ [\max(0, \frac{SR-8}{4})]^2 + [\max(0, \frac{FW-16ML}{12.5ML})]^2$$
$$+ [\max(0, \frac{9ML-FW}{12.5ML})]^2 + [\max(0, \frac{MBS-AIS}{MBS})]^2 \qquad (5\text{-}8)$$

As one can see, the terms of the function are simply the normalized LHS's of the inequality constraints in Figure 5-37. As a rule of thumb for normalization, each term has been divided by the average of its two bracketing terms in the original constraint (if originally a bracketed constraint) or the smaller side of the inequality (if originally an open inequality). For instance, the terms "CD - 0.4" and "0.2 - CD" have been divided by 0.3, which is the average of the bracketing terms in the original constraint *"0.2 m ≤ CD ≤ 0.4 m"*, that is (0.2 + 0.4) / 2 = 0.3. Also, the term "MBS - AIS" has been divided by "MBS" as it is the smaller side of the original inequality *"MBS ≤ AIS"*.

Also, since GAs inherently seek to *maximize* an objective (fitness) function and we are interested in minimizing a penalty function, we define an objective function of the form OF = K - PF where, as explained earlier, K is calculated separately for each generation as the sum of maximum and minimum objective function values in that generation.

The next step would be now to search the space of the problem's free variables. Ranges of values of the four free variables (FW, ML, NPT and AIS) are taken from the respective component cell in components library. Of these variables, the last three are discrete variables[15]. The fourth one, FW, is a continuous variable which we discretize based on the information from a local supplier of gear materials.

Binary codes (substrings) of length 4 are found adequate and are used to represent a maximum of 15 ($=2^4-1$) different discrete values. Therefore, each potential design will be represented by a string of length 16 ($= 4 \times 4$). Figure 5-38 shows the discretized ranges of the four variables. Variables with fewer than 15 discrete values will be given multiples of the same values to have the same number (15) of values and thus to avoid different string lengths. As an example, string (0001 0001 0001 0001) will represent a gear drive with a 1.5

---

[15] This includes those parameters that, although inherently continuous, are normally presented in preferred, discrete values in the literature (e.g. metric gear module). In this example we have adopted the values suggested in(Gieck and Gieck 1990) and (Shigley 1986).

mm module, 15 teeth on its pinion and 12 mm face width made of Cast Iron (ASTM A 48-50 B)[16].

Having specified the solution space (containing $15 \times 15 \times 6 \times 15 = 20250$ points) and the penalty function, now the stage is set for a simple Genetic Algorithm to seek the feasible points. For demonstration purposes, a population size of 12 was selected (instead of a recommended popsize of 17) which in this case performed quite well. As explained earlier, the ratio of population's average fitness to maximum fitness was used to define the termination criterion. The computer code to carry out the search was set to terminate once this ratio exceeded 0.99 (instead of the recommended 0.95, again for demonstrating the convergence accuracy). Other GA parameters were selected as explained earlier in this chapter.

ML(mm)	1.5,	2.0,	2.5,	3.0,	4.0,	5.0	6.0	7.0	8.0	9.0
	10.0,	12.0,	16.0,	20.0	25.0					
NPT	15,	16,	17,	18,	19,	20	21	22	24	26
	28,	30,	34,	38,	40,					
AIS(MPa)[17]	20.0,	20.0,	20.0,	50.0	50.0	50.0	55.0	55.0	57.5	57.5
	72.5	72.5	72.5,	125.0	125.0					
FW(mm)	12.0,	14.0,	16.0,	18.0,	20.0,	25	30	35	40	45
	50.0,	55.0,	60.0,	70.0,	80.0					

Figure 5-38: Discretized ranges of free variables

---

[16] Note that we have chosen not to map the all-zero binary codes. That is the reason why the code "0001" refers to the first value in the domain of a variable. In the computer implementation of the GA, if any of the random operations result in an all-zero code, that operation is repeated until a non-zero code is generated.

[17] In this example, each AIS value uniquely represents a gear material viz (from Gieck and Gieck 1990):

Material	AIS(MPa)
Cast Iron (ASTM A 48-50 B)	20.0
Carbon Steel (ASTM A572 Gr. 65)	50.0
Carbon Steel (ASTM A536 129-90-02)	57.5
Carbon Steel (SAE 1064)	55.0
Alloy Steel (SAE 4140)	72.5
Case Hardened Alloy Steel (SAE 3240)	125.0

Each iteration of the program consisted of finding a "fitter" set of values for the free variables, substituting these values in the equation set, solving it for its output variables (unknowns) and evaluating the penalty/fitness function using values of the entire variable set **x**.

In a typical run of the GA, ten solution alternatives were simultaneously found (Figure 5-39), all of which satisfied the initial requirements and the overall constraint set (C) of the problem. As far as the exploration module is concerned, these alternatives are equally acceptable and are *all* reported to the blackboard of the conceptual designer for further processing.

The evolution of generations of parameter values for this typical run is partially demonstrated in Appendix C. Looking at these results, one can see how fast the "fitter" strings (potential solutions) take over the populations in consecutive generations. One should also note the occasional perturbations in this trend, where weaker strings are generated due to the action of GA operators.

	ML(mm)	NPT	NGT	FW(mm)	AIS(MPa)	MBS(MPa)	CD(m)
alternative 1	6.0	17	68	80.0	72.5	40.11	0.26
alternative 2	6.0	19	76	70.0	72.5	40.26	0.29
alternative 3	4.0	34	136	35.0	125.0	87.21	0.34
alternative 4	6.0	19	76	55.0	125.0	51.23	0.29
alternative 5	6.0	18	72	70.0	72.5	42.87	0.27
alternative 6	6.0	21	84	70.0	125.0	36.01	0.32
alternative 7	5.0	18	72	55.0	72.5	74.34	0.23
alternative 8	6.0	15	60	70.0	72.5	53.44	0.23
alternative 9	6.0	19	76	55.0	72.5	51.23	0.29
alternative 10	6.0	15	60	55.0	72.5	68.01	0.23

Figure 5-39: Feasible solution alternatives

These inferior specimens, however, soon die out and the general trend of improving average fitness continues.

As for the efficiency of the search method, in this particular run the system has started from a totally random set of points and has converged (average fitness of the population within 1% of its maximum fitness) after only 204 (= 17 × 12) points have been examined out of over 20000 points in the solution space. Overall, the program was run 50 times for the same initial specifications. Without using a "rebooting" mechanism, 44 runs converged to the solution. Adding the rebooting mechanism resulted in a 100% success (50 optimal convergences out of 50 runs) for the example problem, all in under 27 iterations (generations).

5.3 THE *EVALUATION* KNOWLEDGE-SOURCE

Remember from Chapter 4 that, at any point during the design process, the common representation database (or the blackboard) of the system exhibits the status quo of the partial designs. The functional representation panel of the blackboard does this by presenting each partial design by its Function Block Diagram. Among the information embedded in the FBD of a design are the Standard Elemental Functions (Chapter 1) that particular design performs, plus the "specifiers" of these SEFs. We defined function specifiers as the performance- (as opposed to design-) parameters that define a function quantitatively, i.e. give us the values by which the function can be "measured". For instance, the elemental function "Change Rotational Speed" has the specifiers "Input or Output Speed", "Speed Ratio" and "Transmitted Power".

We also remember that once a candidate component is "explored" by the exploration module (section 5.2), specifications of the resulting component-instances are reported back to the blackboard where they are used to update the corresponding partial design and hence the search tree. This is where the *evaluation KS* comes into play. Upon activation by the scheduler (design manager), the evaluation KS gets the functional descriptions of both the partial design and the explored component's instance(s). Then, for each feasible instance, it sets up a new leaf node in the search tree, representing one updated partial design. The operation of the evaluation knowledge-source is demonstrated graphically in Figure 5-40.

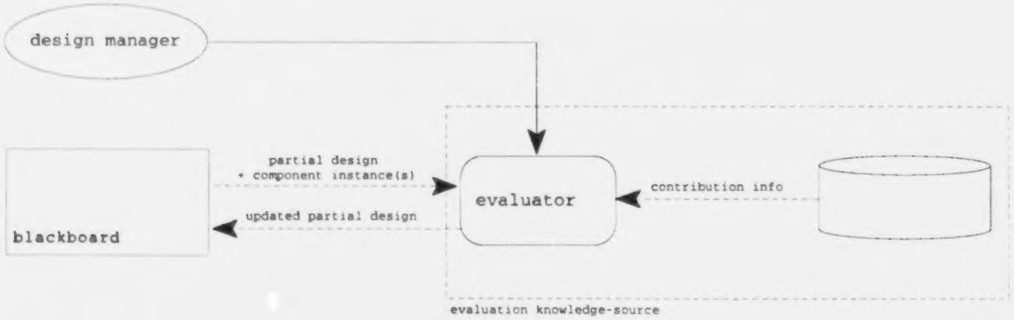


Figure 5-40: The *evaluation* knowledge-source

In order to update each partial design (represented by a node in the search tree) into a number (≥ 1) of new, augmented designs, the KS proceeds as follows (Figure5-41).

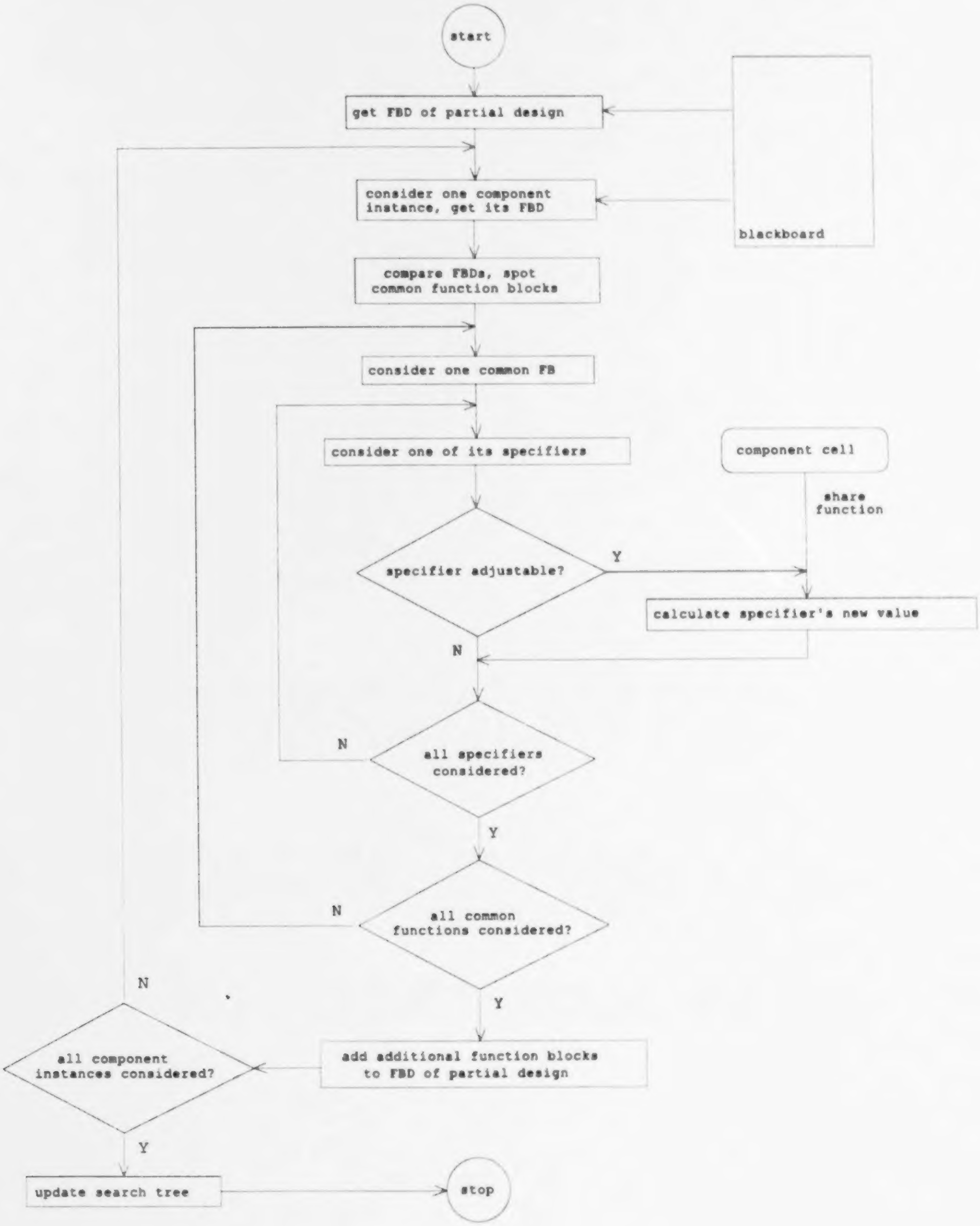

start
get FBD of partial design
blackboard
consider one component instance, get its FBD
compare FBDs, spot common function blocks
consider one common FB
consider one of its specifiers
component cell
share function
specifier adjustable?
Y
calculate specifier's new value
N
N
all specifiers considered?
Y
N
all common functions considered?
Y
N
all component instances considered?
add additional function blocks to FBD of partial design
Y
update search tree
stop

Figure 5-41: The operation of *evaluation* knowledge-source

1- Reads the list of elemental functions and their specifiers pertaining to both the partial design and the component instance just reported in by the exploration module.

2- Compares the "functions" parts of the two lists, spots their common functions and forms the two disjoint subsets "common functions" and "functions in component's list and not in partial design's". If we let F_{PD} denote the set of functions in partial design's list and F_{CI} to denote those in component-instance's list, then the two subsets we are after would be defined as

$$F_{CF} = F_{PD} \cap F_{CI}$$
$$F_{CO} = \{ f : f \in F_{CI}, f \notin F_{CF} \}$$

where F_{CF} denotes the subset of all common functions, F_{CO} denotes the subset of the functions appearing in component-instance's list and not the partial design's, and f denotes a single function in the latter.

3- For each common function, considers the specifiers one by one and checks their "status". (Remember from our discussion of the contents of component-cells in section 5.1 that each specifier parameter is either *adjustable* or *non-adjustable* (Figure 5-2). An adjustable specifier is one that does change due to the contribution of other, similar functions and a non-adjustable specifier is one which does not. For example, the Speed Ratio of a gear drive *is* adjustable as the addition of a second gear drive will result in a new, overall Speed Ratio equal to the product of the first and second ratios, whereas the Rotational Speed of a gear is *not* adjustable by the same token.)

4- For each adjustable specifier of a function, reads the respective *Share Function* (Figure 5-2) from the component cell. Using this function, calculates the adjusted value of the specifier. If the adjusted value equals the initial, desired value (meaning that the function has been carried out to the desired extent), the corresponding function block is removed from the requirements FBD[18]. Otherwise, the new (remaining) value of the specifier is calculated and substituted for the initial value in a copy of the requirements' FBD. The procedure is repeated for all adjustable specifiers of the elemental function, and corrections are made to the same copy of the requirements FBD accordingly. In the end, this updated copy of the requirements' FBD would represent the new (remaining) set of design requirements.

---

[18] In the presentation that follows, we shall frequently use to the terms *requirements' Function block Diagram (FBD)* and *partial design's FBD*. From chapter 3 we remember that the former is a graphical representation of the remaining (unsatisfied) requirements of the problem and the latter is a graphical representation of the functional behavior of a (partial) design at the current stage of problem solving. A third item, namely the *partial design* itself, is also used to complete the description of a node in the search tree. The term *partial design* here refers to the (graphical) structural description of the actual partial design at this stage.

5- In the meantime, adds the same common function block to a copy of the partial design's FBD. This copy, when completed, will "functionally" represent one upgraded partial design.

6- After considering all common functions in steps 4 and 5 above, adds to the newly upgraded FBD of the partial design, all the functions in F_{CO}, that is, all the additional, unintended functions the component performs besides the one(s) for which it was originally picked. While being added to the FBD of the partial design, the additional functions are marked "extra" to indicate that they were not part of the initial requirements of the problem.

7- Updates the search tree by adding to the current node, all the newly generated nodes each representing an upgraded (partial) design. (Naturally there will be as many of them as the number of feasible component-instances reported in by the exploration module.)

Having completed the steps above, the leaf-nodes of the updated search tree now display *tentatively* the current solution state. Each leaf-node, representing an upgraded partial design, now tells us what portions of the initial requirements that design "covers" and which portions are still left to be done, or in brief, how close to a complete design we are. We emphasize the tentativeness of these nodes at this stage because they have to be "verified" before being accepted as actual partial designs. In the next section we shall discuss the verification process and explain how the undesired alternatives are detected to be removed from the search tree.

Let us now illustrate the procedure just outlined in the context of the familiar "drill" example from chapter 3. This time around, we change the figures slightly in order to be able to use the results of our other example, the gear design problem, of this chapter. The problem is shown by its requirements FBD in Figure 5-42: we wish to use a source of mechanical energy (10 kw at1600 rpm) to build a drill whose tip runs at 200 rpm and whose output axis is perpendicular to its input's. These requirements are presented in Figure 5-42 by a set of function blocks each containing an elemental function and its function specifiers.

Now consider the problem at an intermediate solution state in which the first function block has been taken care of, that is, the system has found a number of components to carry out the first elemental function. Of these components, we consider one here. Figure 5-43(a) shows the updated requirements FBD of the problem at this stage. One of the components found by the system for the first function block is shown in Figure 5-43(b) by its FBD. The component is an electric motor whose selection has set up a new node in the search tree

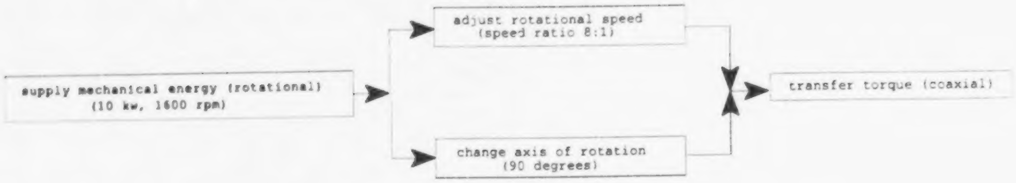
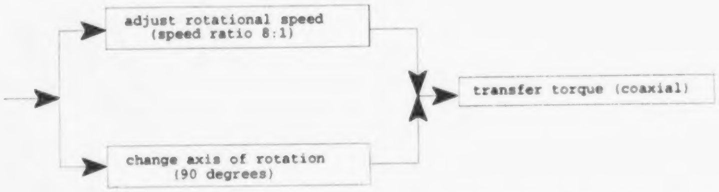

supply mechanical energy (rotational)
(10 kw, 1600 rpm)
adjust rotational speed
(speed ratio 8:1)
change axis of rotation
(90 degrees)
transfer torque (coaxial)

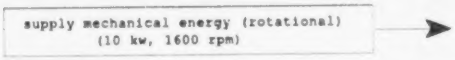
Figure 5-42: Drill design problem, presented by its requirements FBD

(Figure 5-43(c)).

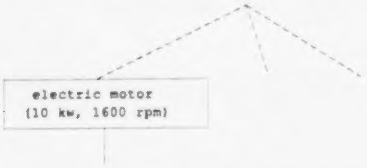
Suppose that in the next iteration, the system picks *spur gear drive* as a candidate component and that present in the problem is a space constraint which requires the speed ratio of a single gear drive not to exceed a value of 4. Also suppose that the products of exploration of this candidate component are the 10 gear-drive instances presented in Figure 5-39, all satisfying the speed ratio constraint. This being the case, the evaluation knowledge-source now takes over and updates the solution state as follows (procedure described for one component instance only, say the first alternative in Figure 5-39; the step numbers do not necessarily match with those presented earlier in this section).


adjust rotational speed
(speed ratio 8:1)
change axis of rotation
(90 degrees)
transfer torque (coaxial)

(a): Updated (remaining) requirements'Function Block Diagram


supply mechanical energy (rotational)
(10 kw, 1600 rpm)

(b): FBD of the electric motor


electric motor
(10 kw, 1600 rpm)

(c): Search tree at this solution state

Figure 5-43: Drill design problem: an intermediate solution state

step 1. Get the current requirements' FBD, Figure 5-43(a), and the FBD of the component instance (Figure 5-44).

step 2. Compare the two FBDs and spot the common function blocks. The only common block between the two is that of "adjust rotational speed".

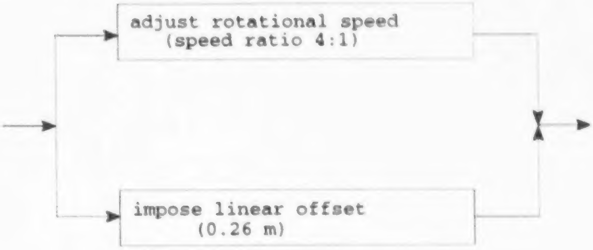


Figure 5-44: FBD of the gear-drive instance

step 3. Determine the adjustable specifiers of the common function and calculate their new values. Of the three function specifiers (Transmitted Power, Speed Ratio and Input/Output Speed) of the above elemental function, only one (Speed Ratio) is adjustable (Figure 5-2). The required value of this specifier is 8 whereas the gear-drive instance at hand can only provide a ratio of 4. Therefore the elemental function is not completely carried out and the new required value of this specifier must be calculated. Using the corresponding share-function from respective component-cell (Figure 5-2) the new value for the Speed Ratio is calculated as

$$SR_N = SR_I / SR_C$$

where

SR_N = New Speed Ratio
SR_I = Initial Speed Ratio
SR_C = Component's Speed Ratio

or $SR_N = 8 / 4 = 2$

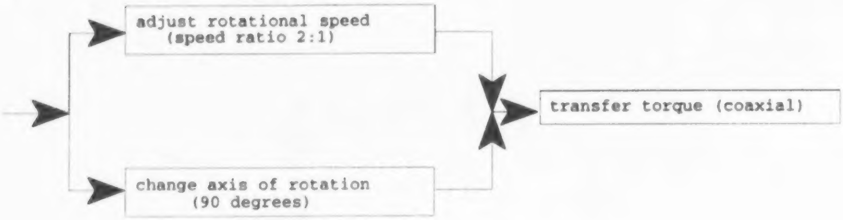
step 4. Make a copy of the requirements' FBD, Figure 5-43(a), and change its specifier "SR" from 8 to 2 so that the modified graph now reflects the remaining (unsatisfied) requirements. The resulting diagram is shown in Figure 5-45(a).

step 5. Add to the partial design's FBD, Figure 5-43(b), the common function block (rotational speed change) from the FBD of the component instance (Figure 5-44).

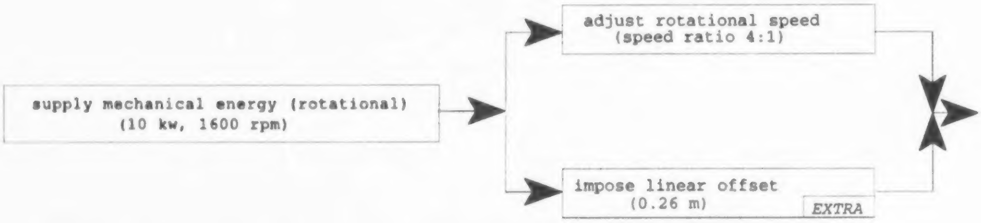
step 6. Add the extra functions the component instance performs, to the upgraded FBD of the partial design generated in step 5 above and label it "extra". In this case we find only one such extra function and that is "linear offset" which refers to the unintended existence of the centre distance of the mating gears. The upgraded partial design's FBD is shown in Figure 5-45(b).

step 7. Update the search tree by adding a leaf-node representing the just-upgraded partial design (Figure 5-45(c)).

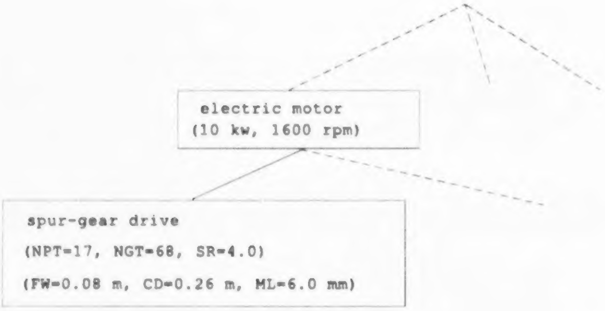
The above procedure is repeated for all 10 instances of the gear drive in Figure 5-39. As a result, 10 new leaf nodes, representing 10 upgraded, yet tentative, partial designs are added to the search tree. The final product of this iteration of the system will be determined at the end of the next stage, namely the verification of the tentative designs.


adjust rotational speed
(speed ratio 2:1)
transfer torque (coaxial)
change axis of rotation
(90 degrees)

(a): Updated (remaining) requirements' Function Block Diagram


adjust rotational speed
(speed ratio 4:1)
supply mechanical energy (rotational)
(10 kw, 1600 rpm)
impose linear offset
(0.26 m)
EXTRA

(b): Updated FBD of the partial design


electric motor
(10 kw, 1600 rpm)
spur-gear drive
(NPT=17, NGT=68, SR=4.0)
(FW=0.08 m, CD=0.26 m, ML=6.0 mm)

(c): Updated earch tree

Figure 5-45: Drill design problem: one iteration later

One final point in this discussion is the fate of the "extra" functions resulting from the unintended activities of the selected components. Two possibilities exist. The extra functions may be negated by the secondary functions of other components in the system, or they may not. For example, the selection of a second bevel-gear set may correct the unwanted offset between the two shafts imposed by a first set. In either case, the *conceptual designer* will not attempt to remove or alter the extra functions if they are not corrected internally. It would rather just label them and keep them in its final report to the user. It is up to him/her then to decide whether the extra functions are harmless to the intended functioning of a design or whether they would render it unacceptable.

5.4 THE *VERIFICATION* KNOWLEDGE-SOURCE

This is the finishing step in the four-step prescription of Design by Exploration. It completes one iteration of the DbE's search scheme for finding multiple feasible solutions to the given problem. The verification KS basically checks the validity of the tentative design alternatives generated and added to the search tree by the other three generating knowledge-sources (namely the nomination KS, the exploration KS and the evaluation KS) against the "precedence" requirements embedded in the initial presentation of the problem.

The verification step is essential to the acceptability of an otherwise satisfactory design because the precedence order of the functions itself is an inseparable part of any functional representation of a design. A design cannot be represented functionally by a bunch of functions related to each other erratically. In chapter one we explained how the configuration of function blocks in a function block diagram implies special relationships (such as control, feedback and input/output) between the corresponding functions.

Verification, as used in this work, is also a means for preventing a combinatorial explosion of the search tree. It works as a "pruning" tool which disqualifies and removes those design alternatives that do not comply with the precedence order requirements. In order to describe the operation of the knowledge-source, we first need to point out a few more details of our "design representation method" which we did not discuss in chapter 1 where the method was initially introduced. We felt that these details would make more sense if we postponed their discussion to this stage where their usage is explained.

The precedence order of the function blocks in an FBD is expressed by a set of predicate logic statements using three basic predicates "follows", "next_to" and "simultaneous". The formats and the application of these predicates is explained below.

- *[Follows (FB_2, FB_1)]* indicates that the function block *"FB_2"* comes somewhere after the function block *"FB_1"* in the diagram. The predicate implies that the occurrence of its first-argument function *"FB_2"* somehow depends on the occurrence of its second-argument function *"FB_1"*. This could be because the former uses the output of the latter as its input (e.g. a pneumatic cylinder uses the compressed air from a compressor), or simply because it is more efficient to do so (e.g. positioning speed reductions in a transmission system as close to the output point as possible would save us thick shafts to carry higher torques).

- *[Next_to (FB_2, FB_1)]* means that the function block "FB_2" comes *immediately* after the function block "FB_1" in the diagram, i.e. no other functions may take place between the two functions. For example, in an internal combustion engine the fuel injection or the spark has to occur exactly at the end of the compression stroke and cannot be postponed till after some other functions of the engine have taken place.
- *[Simultaneous (FB_2, FB_1)]* means the two function blocks FB_1 *and* FB_2 must occur at the same time. For example, in a lathe the rotation of the workpiece and the axial and latteral motions of the cutting tool have to occur at the same time.

Those functions that are not related by one of the above statements may take place at any order relative to each other. For instance, in the drill example presented earlier, we really do not care if the functions "adjust rotational speed" and "change axis of rotation" occur at the same time or if one precedes the other.

To further clarify these definitions, let us consider the hypothetical FBD of Figure 5-46 with the following (given) precedence order.

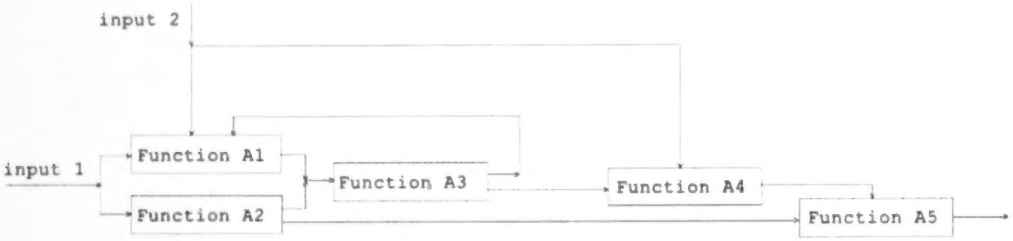


Figure 5-46: A hypothetical FBD

1. *Simultaneous (A1, A2)*
2. *Follows (A3, A1)*
3. *Follows (A3, A2)*
4. *Follows (A4, A3)*
5. *Next_to (A5, A4)*

The first statement indicates that functions A1 and A2 must be carried out simultaneously. The next two statements explain that function A3 must occur after both functions A1 and A2 are accomplished. Similarly, statement 4 indicates that function A4 occurs after A3. Finally, the last statement tells us that function A5 takes place immediately after function A4.

Now suppose that in some stage of the design process, the *conceptual designer* comes up with the two design ideas shown in Figure 5-47. As the figure shows, both alternatives include sought-after function blocks appearing in Figure 5-46 and let us suppose that they have both passed the evaluation stage. However, they are both rejected in the verification stage for the following reasons.

The first design carries out function A5 before function A2. This violates the precedence requirements in two ways. First, functions A1 and A2 are not carried out simultaneously (violation of statement 1). Second, it can be inferred from statements 3 [*Follows (A3, A2)*], 4 [*Follows (A4, A3)*] and 5 [*Next_to (A5, A4)*] that function A5 has to occur *after* function A2, i.e. [*Follows (A5, A2)*]. The first design is therefore disqualified as it violates the precedence requirements. Also the second design is dismissed on the grounds that it interrupts the adjacency of functions A4 and A5 by some function A6.

The operation of the verification KS is presented graphically in Figure 5-48 and is outlined here. For brevity, we shall refer to a function block in the requirements' FBD of a problem as FB1 and to one in the FBD of a (partial) design as FB2.

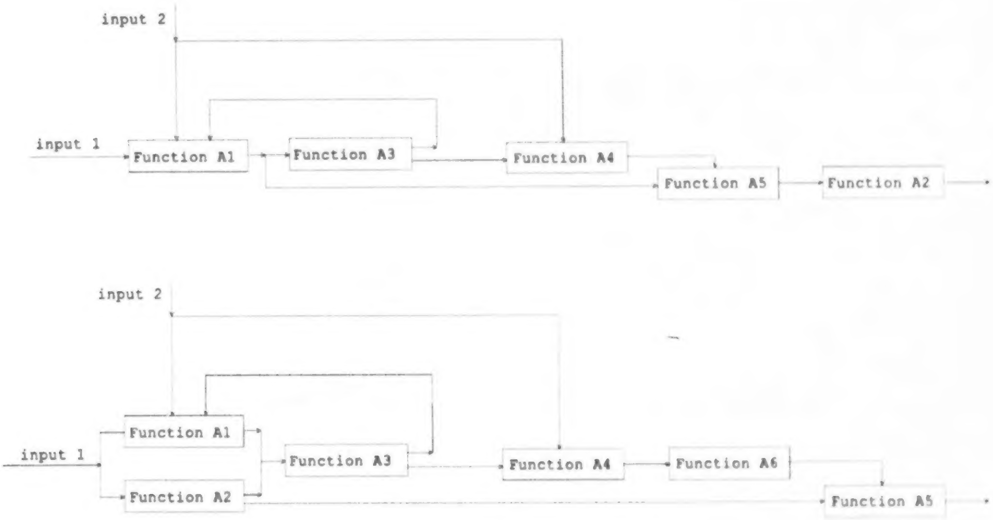


Figure 5-47: Two candidate solutions

- Given the FBD of a generated and evaluated (partial) design, list the function blocks common between this and the requirements FBD of the problem (FB2s and FB1s respectively).

- Spot and isolate all the precedence predicates involving "non-extra" FB2s in the requirements FBD, then
  - * Check the *Next_to* predicates. If an FB2 appears as the first argument of one such predicate, check to see if the second argument is another FB2 occurring immediately before the first one. If not, refute the design and quit.
  - * Check the *Simultaneous* predicates. If an FB2 appears as one of the two arguments in one such predicate, check to see if the other argument is another FB2 occurring simultaneously. If not, refute the design and quit.
  - * Check the *Follows* predicates. If an FB2 appears as the first argument of one such predicate, check to see if the predicate is, directly or indirectly, true with respect to the FBD of the design If not, refute the design and quit.
  - * If the above check OK, endorse the (partial) design and quit.

Note that we do not check for the *Next_to* or the *Follows* predicates that have FB2s as their *second* arguments, as this could wrongfully refute some alternatives in case of partial designs. Such predicate statements, even if not true at one point, might wind up true in succeeding stages of the design process as new components are added to the partial designs. For example, if we had a statement *Follows (function 2, function 1)* and a partial design with a *function 1* but no *function 2* to follow it, it would be wrong to refute it at this stage (before the completion of the design), because chances are that a new component with a *function 2* is added to the partial design at a later stage and satisfies the statement.

This wraps up our discussion of the verification knowledge-source as well as the presentation of strategies and procedures involved in the implementation of the Design by Exploration method.

5.5 SUMMARY AND CONCLUDING REMARKS

In this rather long chapter we elaborated the problem-solving strategies and techniques to implement the ideas presented in chapter 4. According to the DbE model, the conceptual design of a mechanical system is an iterative process involving a sequence of "generate and refine" steps. The model prescribes that, in order to increase the efficiency of the process, generation and refinement phases occur simultaneously and the latter not be postponed till the end of the generation phase. In accordance with this requirement, we asserted that each iteration of the process should involve four basic "generating" steps (nomination, exploration, evaluation and verification) and a possible "criticizing" step. Each of these steps would be carried out by one or more blackboard "knowledge-sources". This

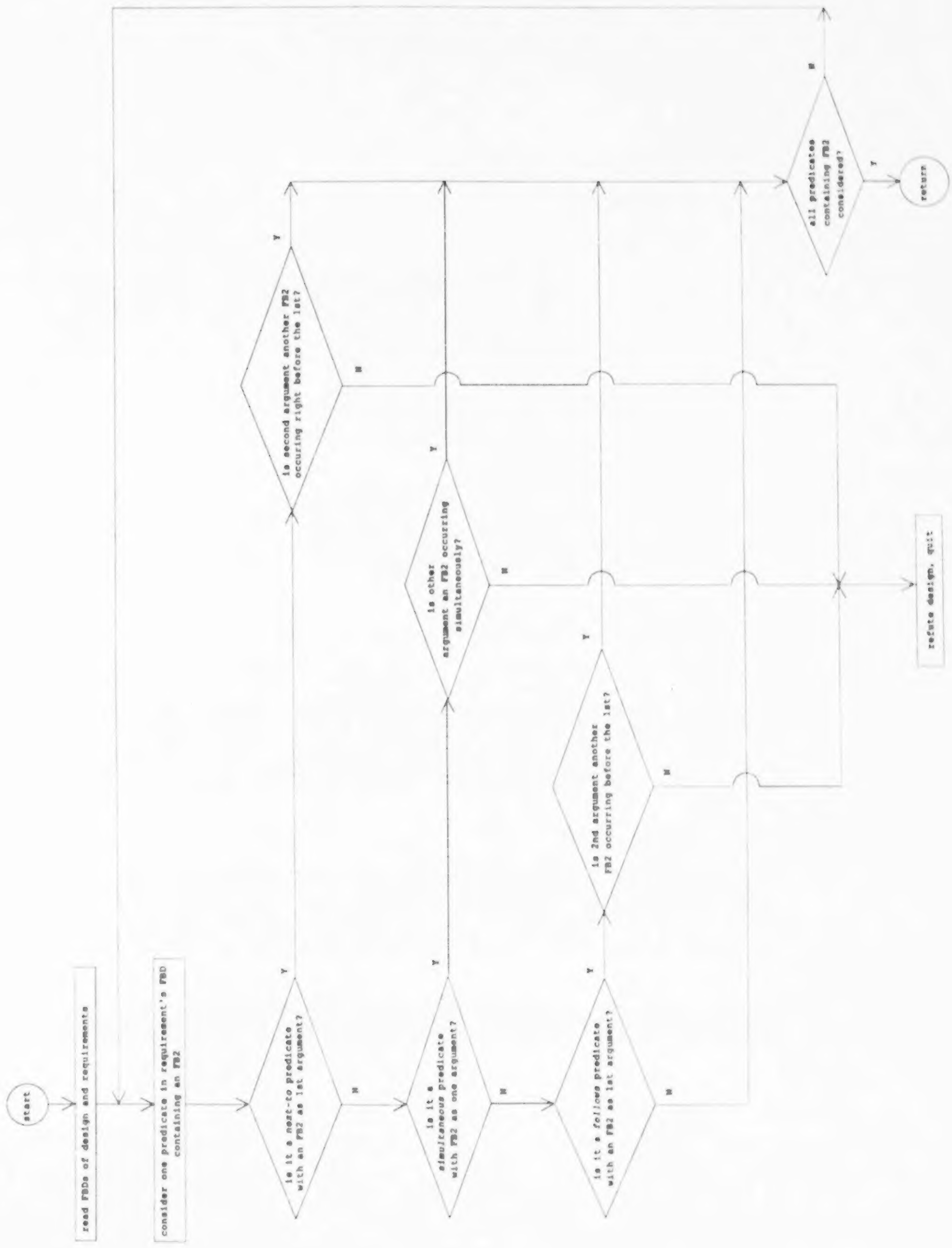

start
read FBDs of design and requirements
consider one predicate in requirement's FBD containing an FB2
is it a *next-to* predicate with an FB2 as 1st argument?
Y
N
is it a *simultaneous* predicate with FB2 as one argument?
Y
N
is it a *follows* predicate with an FB2 as 1st argument?
Y
N
is second argument another FB2 occuring right before the 1st?
Y
N
is other argument an FB2 occurring simultaneously?
Y
N
is 2nd argument another FB2 occurring before the 1st?
Y
N
all predicates containing FB2 considered?
Y
N
return
refute design, quit

Figure 5-48: Operation of *verification* knowledge-source

chapter was devoted to describing the contents and the operation of the four generating knowledge-sources. We explained that

- The *nomination KS* systematically searches a library of component-cells, in the form of individual ".obj" files, to find and "nominate" *all* those components that perform at least one of the required elemental functions of the problem. It uses an index to the function-defining predicate statements of the components to track down those component-cells containing desired functions. The components thus found are called "candidate components".
- The *exploration KS* forms and examines the knowledge-pool of each candidate component to see if it is readily solvable for finite solutions or otherwise if it can be rendered solvable by a reconfiguration of the given information. In either case, it then reformulates the problem and applies to it a case-independent solution technique (genetic search) to find a set of feasible, but not necessarily optimal, solutions. Each solution will then represent an instance of the candidate component, satisfying all the requirements of the problem. The solutions thus found are called tentative solutions (designs) as they are still to be verified against some configurational requirements. As mentioned earlier, the purpose of this step is to fully understand the functional behavior of a promising component so that we can determine its contribution to the satisfaction of the problem's requirements.
- The *evaluation KS* examines the status of the problem after successful exploration of each component and updates the problem's representation so that the new representation would reflect the following.
  - * How does the new component contribute to the fulfilment of the problem's requirements, i.e. what is the functional behavior of the augmented (partial) design?
  - * What else (if at all) should this augmented design do to be considered complete, that is, what functions are still "uncovered" by the design and need to be carried out by the addition of other components.The knowledge-source does this by comparing the function block diagrams of the partial design and the problem requirements, and modifying the latter accordingly. The (partial) designs at this stage are considered *tentative* and are subject to one more check before they can be accepted to the actual family of design alternatives.
- The *verification KS* checks the tentative designs of the previous stages for their "configurations". This is because the feasibility of a (successfully explored) component does not automatically guarantee the approval of the design it is a part of. The KS compares the precedence order of the individual functions in each tentative design with that of the problem requirements and approves only those designs whose

functional configuration matches that of the requirements.

These knowledge-sources (or expert modules), when regarded in the context of the *conceptual designer's* system architecture presented in Chapter 4, provide the means for the automated conceptual design of a class of mechanical systems. These are the systems that can be considered "configurations of existing components", and that have been "functionally designed" already (see the definition in Chapter 1).

Wherever required, we justified our choice of methods while comparing them with others, and tried to provide a brief overview of the more novel methods for the benefit of non-expert readers. We also illustrated the operation of the presented techniques through a number of examples; and reported the results of a series of others designed to help us improve the efficiency and performance of those techniques.

In developing various knowledge-sources we have emphasized those aspects that concern *design automation.* This has always been a challenging task as the design process relies on so many decision making steps. In trying to automate these steps we have been forced to make some assumptions to reduce the otherwise unmanageable complexity of the problem. These include the assumption of all requirements/constraints of the problem being in terms of the design/performance parameters and the assumption that there always exists a discretization mesh to approximate the domains of continuous variables satisfactorily and accurately enough.

One of the other assumptions we have made is that the constraint set of the mechanical components we are dealing with, or their equality constraints to be more accurate, comprises algebraic and/or transcendental functions, and differential equations are not involved. This limitation stems from the fact that there does not exist a "universal" solver capable of solving *all* types of equations and therefore we have to be selective as what class of equations we allow in the problem. To solve our equality constraint sets we currently use "TK Solver Plus" (1988), a mathematical package which uses a modified Newton-Raphson method to solve equation sets, among other functions.

Borland's Turbo PROLOG 2.0 was chosen, for the most part, as the language of implementation of the knowledge-sources because of its modular programming and deductive capabilities. The genetic search of the exploration module was coded in Microsoft's Quick BASIC 4.5 for its string processing abilities.

All the procedures in knowledge-sources (except for some minor parts of the exploration module) have been individually developed in code. The coding of the blackboard, however, is not complete at this time and therefore its functions (including communications between various KSs and the presentation of the design) are being carried out manually.

Also, our library of components which at this stage contains a limited number of component cells needs to be expanded to allow for the composition of more complex systems. The contents of the component cells are taken from (Gieck and Gieck 1990; Shigley and Mischke 1986; Juvinall and Marshek 1991).

In the next chapter we shall present a complete application of the conceptual designer in the context of a comprehensive design example.

CHAPTER 6

APPLICATION EXAMPLE

In this chapter we shall illustrate the performance of the *Design by Exploration* model and its implementation, the *Conceptual Designer,* in the context of a class 2 design problem (Chapter 1). The performance of individual steps of the multi-step model have been already demonstrated through brief examples in respective chapters. Nonetheless, we felt that a complete design example would help the reader grasp the rationales of the model and would enable us to articulate the functioning of the Conceptual Designer as an integrated system. Occasionally in this chapter we shall underline some of the less-obvious vigors of the model as well as the limitations of design system.

Due to the large volume of computations involved in the solution of a problem of this size, we shall not be presenting all the detailed computations for all the steps involved. We will rather present sample computations while making sure the results of every computation/decision-making step is presented in sufficient detail.

Before proceeding to the presentation of the example, here we briefly review the basic steps involved in the implementation of the DbE model one more time. This is necessary because we shall frequently refer to these steps in our discussion of the example problem.

6.1 *DESIGN BY EXPLORATION* REVISITED

Here is an outline of the action cycle of the conceptual designer. The terminology used in the outline is the one presented in chapter 4. In the rest of this chapter, we shall be referring to the following step numbers.

1) Given the definition of the problem in terms of the Standard Elemental Functions, a formal representation of this is posted on the blackboard. This is where an up-to-the-moment report on the functional and structural status of the (partial) designs will be kept throughout the design process. At each point, the report will include the accomplished portion of the requirements as well as the remaining portion. As elaborated earlier, the blackboard comprises a functional representation panel and a structural representation panel.

2) The *design generation* stage begins by searching the system's components-database and finding *all* the promising components, i.e. those that potentially perform at least one of the required Elemental Functions (starting with the first one). These

components are reported back to the blackboard (structural representation panel) where they form the first level of a tree, of which each branch will ultimately represent a solution alternative (this is called the *search tree*).

3) For each selected component, a *knowledge-pool* is formed which contains information from the respective "component cell" (in the system's components-database) and the initial problem representation. This information is basically composed of the component's design equations, constraints and data as well as the initial specifications and constraints[1].

4) The functional behavior of each selected component is "explored" to reveal *all* the functions (intended and unintended) it potentially performs plus the related quantitative information. This is done through examining the component's knowledge pool and solving the corresponding constraint set for the given specifications (for details see Chapter 5). The result will be a number (≥ 1) of feasible instances of the component with the behavior-defining parameters of each instance determined.

5) The functional search tree is updated[2], that is, for each component-instance just explored, a new node is generated by adding the functions of the explored component to the existing functions of the parent node. The resulting set of functions at each node is then compared with the initial requirements of the problem and the new "accomplished" and "remaining" portions of those requirements are determined and specified on the functional-representation panel of the blackboard.

6) Each leaf-node in the updated functional search tree is verified against the initial requirements of the problem. Those nodes that do not comply with the requirements (e.g. violate some of the constraints or do not have the same precedence order among their functions) are discarded. This is called "pruning" the functional serach tree. Each of the remaining nodes of the tree would now represent a candidate (partial) solution.

---

[1] In case of "look-up" components, i.e. the ones that are selected from a catalog, the knowledge-pool will contain specifications tables and/or graphs instead of design equations; and the "exploration" stage (next step) is replaced with a "look-up and validate" procedure whereby an instance of the component is selected which will satisfy applicable constraints (if any).

[2] Note that here we are dealing with two search trees in parallel. One is the *structural search tree* which is displayed on the structural representation panel of the blackboard, and the other is the *functional search tree* which is displayed on the functional representation panel. Each tree is the projection of the other tree in its own panel, meaning that each node of one tree corresponds to a node in the other on a one-to-one basis. For brevity, from now on we shall call the former the *search tree* and the latter the *functional search tree*.

7) Each candidate[3] (partial) design (represented by a branch of the search tree) is examined by any other "expert agent" (or knowledge-source as referred to in the previous chapters) possibly present in the system (such as cost, manufacture or maintenance experts). A candidate design examined by the expert-agent(s) will be endorsed, rejected or modified. The results will be then reported back to the blackboard.

8) The (structural) search tree is updated by mapping the latest level of the functional search tree onto the structural representation panel. In other words, each feasible component-instance (found in step 4 above and passed the validity checks of steps 6 and 7) is added to its parent node in the search tree to form a new leaf-node. The path (or the branch) leading to a leaf-node will then represent a candidate alternative (partial) design *in terms of its physical components.*

9) The "surviving" leaf-nodes of the search tree now represent the *feasible* partial/complete solutions. Complete designs are now presented to the user and, unless he/she decides to terminate the process, the system proceeds by repeating steps 2 to 8 above for the partial designs in the search tree. The cycle is repeated until all branches of the search tree represent complete designs or until the system/user decides that no further improvement is possible/needed.

Having outlined the *conceptual design* process as prescribed by the DbE model, we will now proceed with the presentation of the example problem.

6.2 AN ELEVATOR FOR THE DISABLED

The problem is to design the drive system of an elevator for the disabled. The elevator is to be installed in an existing house and is to carry one person in a wheelchair accompanied by a second person. It will travel a maximum of 16 feet (from basement to second floor) at a speed of 0.67 ft/sec (16 feet in 24 seconds).

This problem was chosen because it was also assigned to several groups of senior undergraduate students at the mechanical engineering department of the university of Alberta as their final design project. Therefore, a comparison can be made between the results of DbE and those of human designers, albeit designers with minimal experience. In what follows, the performance and results of DbE will be explored and then the ideas generated by the students will be briefly presented and comparisons between the two will be made.

Since the purpose of this example is only to illustrate the functioning of the design system, wherever possible we have allowed some simplifications in the actual design problem to avoid unnecessary complications.

---

[3] The word *candidate* here refers to the fact that any partial design generated to this point may be subject to other validity checks by various critic knowledge-sources as discussed in the previous chapters. Only after passing these possible ckecks will a candidate design be considered a (partial) design.

6.2.1 INITIAL SPECIFICATIONS, ASSUMPTIONS AND CONSTRAINTS

The total weight of the passengers is given to be 200 kgf. The empty weight of the elevator box is also 200 kgf. The total weight of the system is the sum of these two values or 400 kgf. According to (Kogan 1985), an overload coefficient of 1.3 should be considered in calculations. This brings the total weight of the system to 520 kgf or 1150 lbf approximately[4].

The elevator travels between the basement and the second floor. It does this at a rate of one floor per 12 seconds. Assuming a uniform speed (no slow downs), this means a speed of 0.67 ft/sec or 40 ft/min. The required power to lift the elevator can be calculated as follows.

$$\textit{required power (hp)} = \frac{\textit{load (lb)} * \textit{velocity (ft/}\mathrm{min})}{33000 * \textit{drive system efficiency}} \tag{6-1}$$

For the information given above and an average efficiency of 67.5% for various types of elevators (Strakosch 1983) we get[5]:

$$\textit{required power} = \frac{1150\ lb * 40\ ft/\mathrm{min}}{33000 * 0.675} \approx 2\ hp$$

6.3 INITIAL REPRESENTATION OF THE PROBLEM

The Function Block Diagram representation of the initial requirements is shown in Figure 6-1. The diagram uses Standard Elemental Functions (Chapter 1) to represent the set of actions to be taken and the order in which they are to be carried out. The function blocks have been numbered for later references. According to the diagram, in the most general case the following should be carried out.

- On passenger's signal (e.g. push of a button), a source of mechanical energy (with a minimum power of 2.0 hp) starts delivering (generating) motion (linear or rotational);

---

[4] In order to be consistent with the catalogs, handbooks and standards available to us, we have decided to use the imperial system of units throughout this chapter.

[5] Note that for the purpose of uniformity we have overlooked the effect of a *counterweight* used in some types of elevators to reduce the required power.

- If required, the speed is adjusted to suit the next function. In this case an input speed (V_1, determined by function 1) would have to be changed to an output speed (V_2, determined by function 3). Having mapped function 1, the design system would map function 3 and check the resulting component(s) for a possible input-speed constraint. If one exists that is violated by the speed available at the end of function 1 it would back up one step and map function 2.
- If not already, the motion is converted to a linear one which will be then transferred to the elevator box. This would either have the desired speed of 0.67 ft/sec or some other speed which can later be adjusted to the desired value.
- If required, the linear speed is adjusted to the desired value (0.67 ft/sec);
- To provide for the desired stops, the position of the elevator box is sensed. As soon as the box is at a requested stop point, the source of motion is turned off and the system is no more powered;
- To provide for accurate stops, a brake is applied to the system to stop the possible, inertia-related motion of the box;
- As a safety measure against the free drop of the elevator box, the acceleration of the box is continually measured. If it exceeds a prescribed value (e.g. that of gravity), the source of energy is immediately disconnected and the box is "grabbed" and held stationary.

The order in which these functions are to be performed is as follows.

- Function 2, if required, is to immediately follow function 1;
- Similarly, function 3 is to occur right after function 2;
- Function 4 may be performed anywhere after function 3;
- Functions 6 and 7 are to be carried out simultaneously;
- Functions 9 and 10 are to be carried out simultaneously.

In this representation of the problem note that:

- The three branches of the diagram are joined by an "and" joint, meaning that they are to be performed in parallel (Chapter 1);
- The diagram represents the problem in its most general case with minimum favoritism towards a particular solution. For instance, "speed adjustment" has not been specifically placed before or after "motion linearization" to favor a geared- or a gear-less drive system respectively.
- The function blocks are "component-neutral" in the sense that they do not specifically refer to particular components/subsystems. For example, the function "adjust speed" could be equally mapped to more than a dozen mechanisms

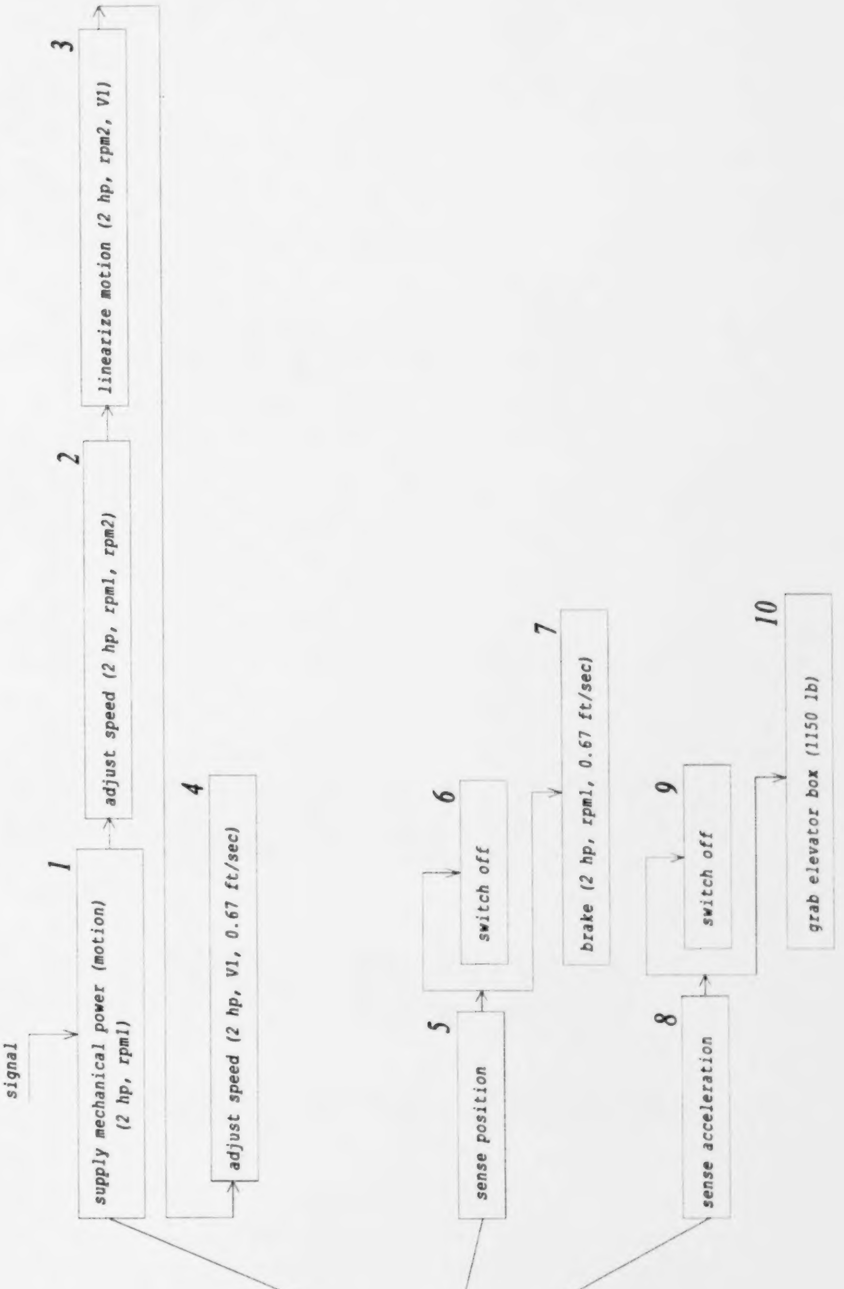

signal
1
supply mechanical power (motion) (2 hp, rpm1)
2
adjust speed (2 hp, rpm1, rpm2)
3
linearize motion (2 hp, rpm2, V1)
4
adjust speed (2 hp, V1, 0.67 ft/sec)
5
sense position
6
switch off
7
brake (2 hp, rpm1, 0.67 ft/sec)
8
sense acceleration
9
switch off
10
grab elevator box (1150 lb)
and

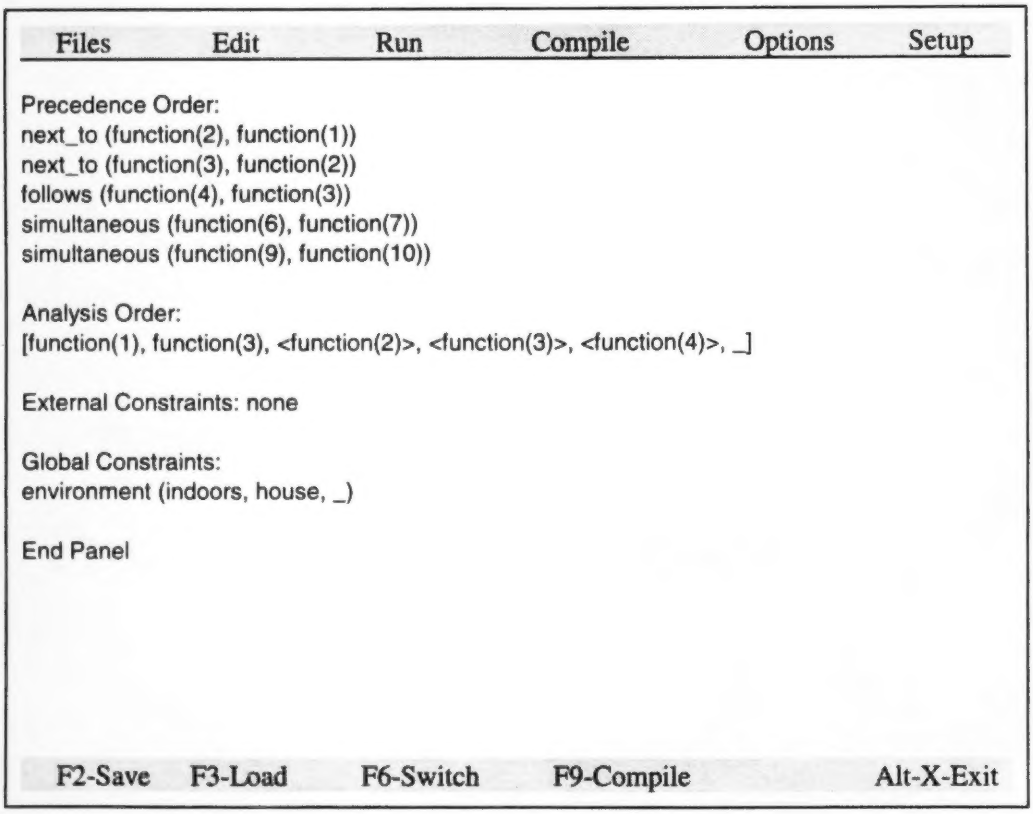
Figure 6-1: Functional description of the elevator drive system

including gear sets and belt-/chain drives.

- As it is commonly true of the real-world problems, the problem is highly underspecified in terms of the number of design variables initially specified. Many more variables need to be specified before the problem can be solved by traditional design methods (chapter 1). As discussed earlier, special computational techniques are needed here to replace the "experience" and the "intuition" that computers lack.

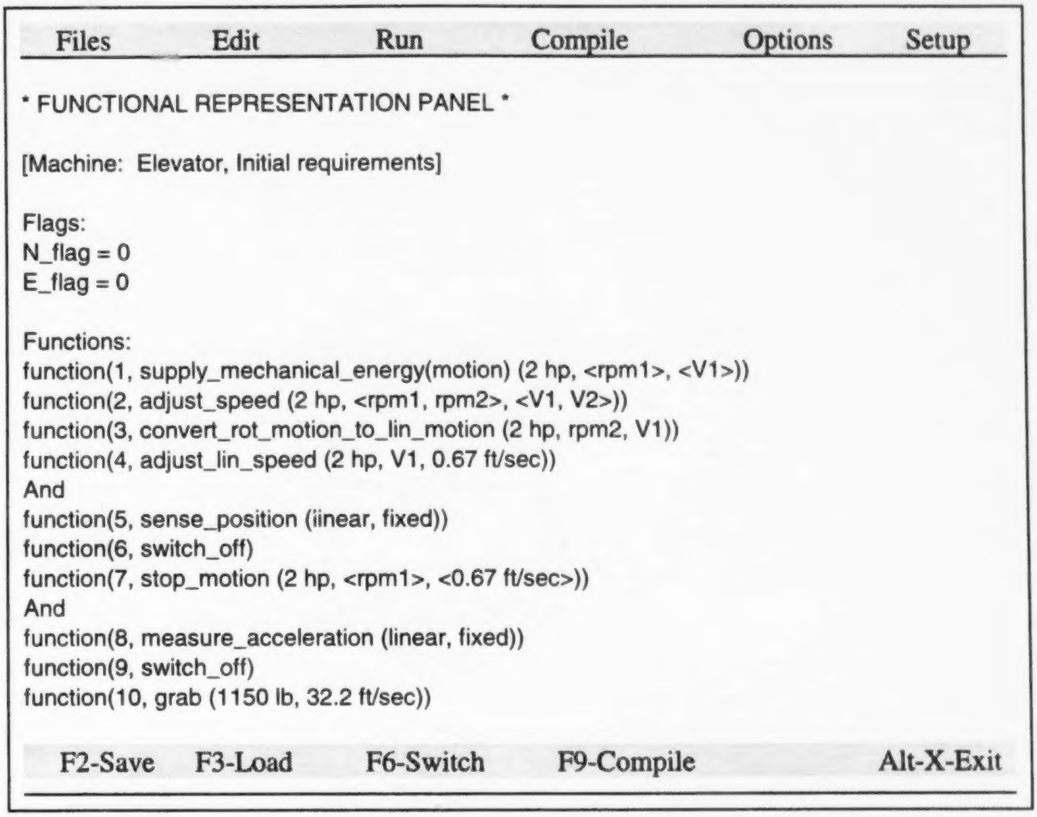
6.4 FORMAL REPRESENTATION OF THE PROBLEM (STEP 1)

Figure 6-2 shows the functional representation panel of the *conceptual designer*'s blackboard. The panel, shown in Turbo Prolog's dialog window, contains the predicate representation of the problem's functional requirements (Figure 6-1) plus the applicable (external and global) constraints and a number of system *flags* (see Chapter 4 for definitions). Note that in this case the only constraint included in the problem is that the elevator will be



```
Precedence Order:
next_to (function(2), function(1))
next_to (function(3), function(2))
follows (function(4), function(3))
simultaneous (function(6), function(7))
simultaneous (function(9), function(10))

Analysis Order:
[function(1), function(3), <function(2)>, <function(3)>, <function(4)>, _]

External Constraints: none

Global Constraints:
environment (indoors, house, _)

End Panel
```

Figure 6-2: Formal representation of the initial requirements



```
* FUNCTIONAL REPRESENTATION PANEL *

[Machine:  Elevator, Initial requirements]

Flags:
N_flag = 0
E_flag = 0

Functions:
function(1, supply_mechanical_energy(motion) (2 hp, <rpm1>, <V1>))
function(2, adjust_speed (2 hp, <rpm1, rpm2>, <V1, V2>))
function(3, convert_rot_motion_to_lin_motion (2 hp, rpm2, V1))
function(4, adjust_lin_speed (2 hp, V1, 0.67 ft/sec))
And
function(5, sense_position (iinear, fixed))
function(6, switch_off)
function(7, stop_motion (2 hp, <rpm1>, <0.67 ft/sec>))
And
function(8, measure_acceleration (linear, fixed))
function(9, switch_off)
function(10, grab (1150 lb, 32.2 ft/sec))
```

Figure 6-2 (continued)

used in an existing house. Later on we shall see how an environment knowledge-source will check the candidate designs for the satisfaction of this constraint.

In the representation, a pair of brackets (<>) indicates the "optional" nature of a function or its specifiers. It can specify multiple options (in the case of function specifiers) or denote the possibility of an elemental function not being implemented (as indicated by the "analysis order" section of the problem representation). This will be further articulated in the following sections.

6.5 GENERATING CANDIDATE DESIGNS

6.5.1 NOMINATING CANDIDATE COMPONENTS (STEP 2)

Starting with function block 1 (supply mechanical energy), the Conceptual Designer scans its components-database in order to find all those components that perform (at least) the specified function. An index to the database (Appendix D) in which each entry corresponds to an individual component-cell allows the system to spot desired components by matching the specified function against the "functions" part of each record. In this case, the system finds four candidate component-types[6] with the specified function in their "resume". These components are "electric motors", "electric gear-motors", "electric brake-motors" and "internal combustion engines".

Note that although the first three components are members of the same family (electric motors), they differ in the functions they perform; and since conceptual design primarily revolves around "functions", the Conceptual Designer considers those components separately. The implications of this decision will be articulated as we go on.

The results of the search are posted on the blackboard (Figure 6-3). As outlined in step 2, each of the four components becomes a leaf-node in a search-tree. Then according to the definition of a (candidate) design alternative (Chapter 4, also step 7 above), we have four candidate alternatives, each comprising a single component/subsystem thus far. In Figure 6-3 these alternatives are numbered from 1 to 4. Conceptual Designer uses a local numbering system, meaning that each time the search tree is updated, the alternatives at the lowest tree-level are re-numbered.

In the records shown, an (S) indicates that the alternatives are Structural (as opposed to Functional) and the subscript (ten) refers to the fact that these tree-nodes are tentative in the sense that actual nodes correspond to component-instances rather than component-types, as we shall discuss later.

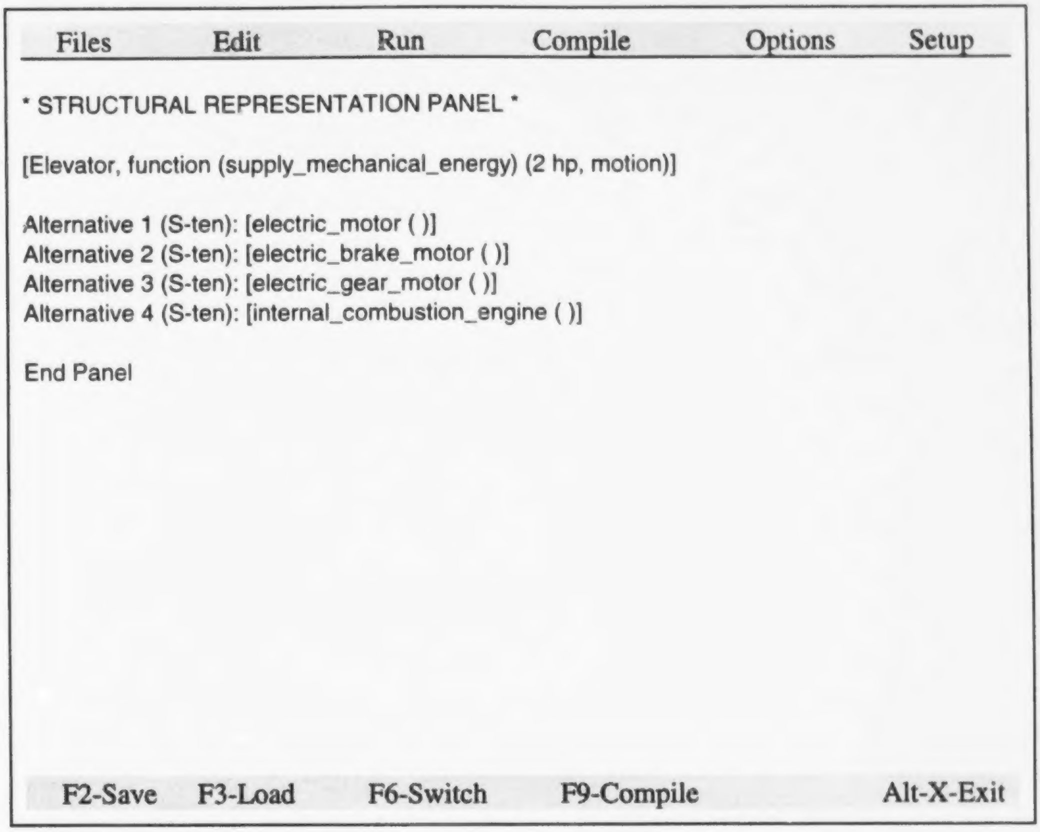
6.5.2 EXPLORING CANDIDATE COMPONENTS

In this particular problem, all four components initially selected are "look-up" components, that is, their "exploration" involves fetching the specifications of the feasible component instances from specifications-tables stored in corresponding component-cells. Since in this chapter we intend to demonstrate the more innovative aspects of the DbE model, and since look-up procedures do not particularly serve this purpose, we will skip details of steps 3 and 4 in this iteration and will only present the results of these steps. The exploration stage will be demonstrated in detail in the next iteration of the design process.

Figure 6-4 shows the outcome from steps 3 and 4 above. Conceptual Designer has been able to find 3 instances of electric motors (Leeson Catalog), 1 instance of brake motors

---

[6] As explained earlier, the term *component-type* refers to a generic component / family of components whose specifications have not been determined yet. Once these specifications are determined, the term *component instance* is used instead. For brevity, however, we shall occasionally use the term *component* to refer to either case (only when the context prevents confusion).



* STRUCTURAL REPRESENTATION PANEL *

[Elevator, function (supply_mechanical_energy) (2 hp, motion)]

Alternative 1 (S-ten): [electric_motor ( )]
Alternative 2 (S-ten): [electric_brake_motor ( )]
Alternative 3 (S-ten): [electric_gear_motor ( )]
Alternative 4 (S-ten): [internal_combustion_engine ( )]

End Panel

Figure 6-3: Nominated components to perform the first elemental function

(Leeson Catalog), 9 instances of gear-motors (FMC Catalog) and 1 instance of i. c. engines[7] that match the stated requirements, i.e. can generate mechanical energy at a rate of 2 hp in the form of motion (in this case rotational) as stated by the first elemental function.

The exploration of the nominated components has revealed the additional functions each component potentially performs besides the requested one. Gear motors, for example, not only supply rotational motion but also adjust their initial output speed with various ratios. Each record in Figure 6-4 includes the catalog number of a selected component instance as well as the names and the specifiers of the functions it performs.

In these records an "F" indicates that the corresponding component is a node in the Functional search tree. Also, the index "can" refers to the fact that at this stage the

---

[7] Since a small-engines catalog was not available at the time, the specifications of a two-cylinder engine were manually added to the database for demonstration purposes.

Alternative 1 (F-can): [electric_motor (110362, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 3450 rpm)]

Alternative 2 (F-can): [electric_motor (120045, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1740 rpm)]

Alternative 3 (F-can): [electric_motor (120055, TP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1740 rpm)]

Alternative 4 (F-can): [electric_brake_motor (12052822, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1740 rpm), "stop_rotational_motion"(2 hp, 1740 rpm)]

Alternative 5 (F-can): [electric_gear_motor (2ADBL2450, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 280 rpm)]

Alternative 6 (F-can): [electric_gear_motor (2ADBL2663, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 188 rpm)]

Alternative 7 (F-can): [electric_gear_motor (2ADB21010, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 125 rpm)]

Alternative 8 (F-can): [electric_gear_motor (2BDBL21500, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 84 rpm)]

Alternative 9 (F-can): [electric_gear_motor (2CDB22800, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 45 rpm)]

Alternative 10 (F-can): [electric_gear_motor (2CTB24200, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 30 rpm)]

Alternative 11 (F-can): [electric_gear_motor (2DTB27640, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 16 rpm)]

Alternative 12 (F-can): [electric_gear_motor (2ETB214000, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 9 rpm)]

Alternative 13 (F-can): [electric_gear_motor (2ETB216800, SP), "supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm), "adjust_rotational_speed"(2 hp, 1750 rpm, 7.5 rpm)]

Alternative 14 (F-can) [internal_combustion_engine (2 cyl), "supply_mechanical_energy(rot. mot.)" (2 hp, 1500 rpm), "convert_chem_energy_to_mech_energy"]

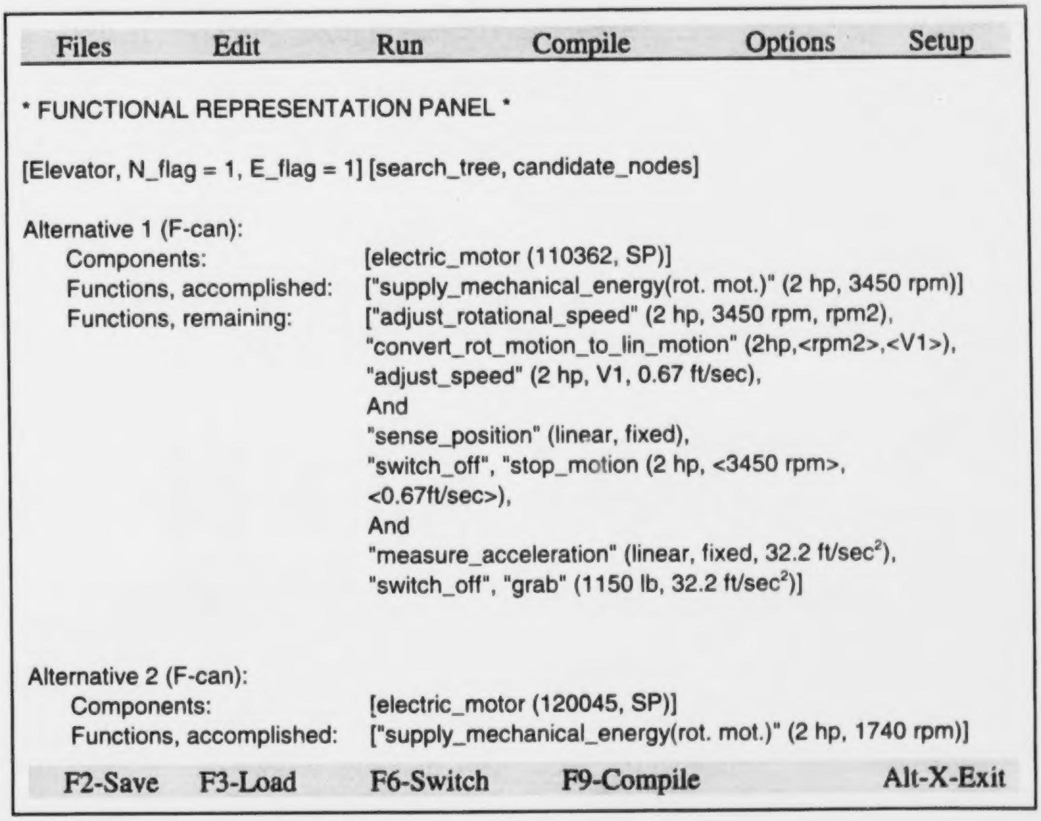
Figure 6-4: Feasible instances of the candidate components - first level

components are only "candidates". Other abbreviations include "SP" standing for "Single Phase" and "TP" standing for "Three Phase" motors.

6.5.3 UPDATING THE FUNCTIONAL SEARCH TREE (STEP 5)

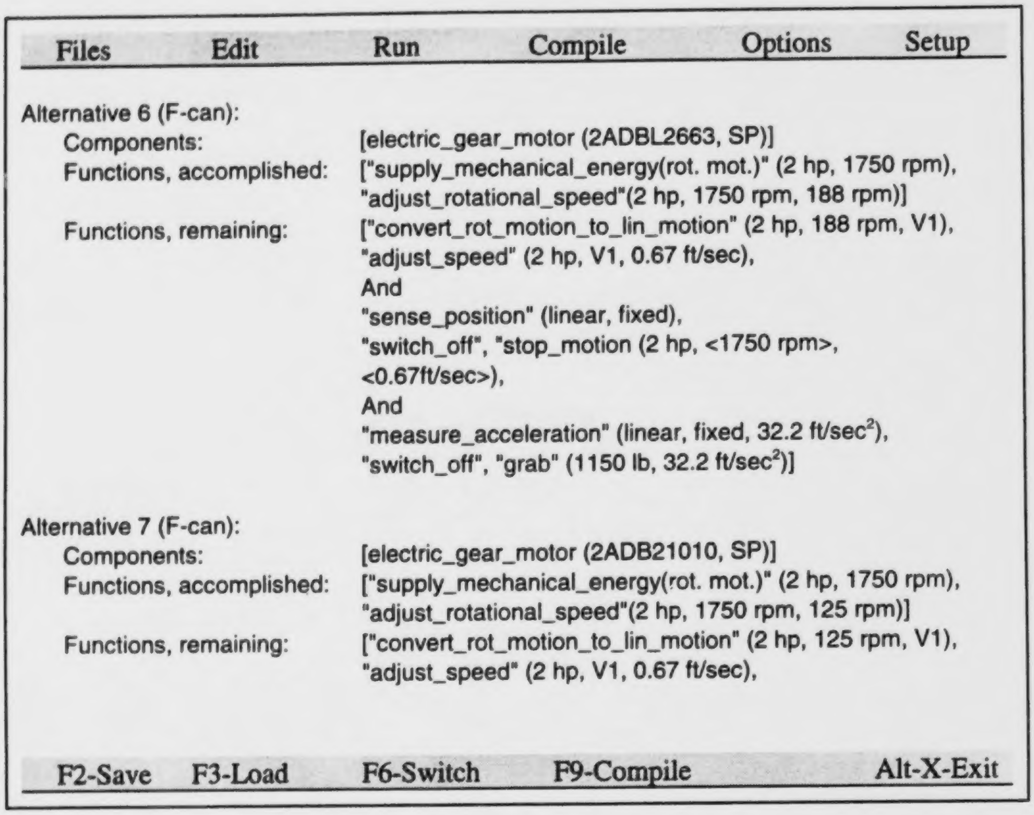
Each of the component instances presented in Figure 6-4 is now mounted as a leaf-node in the functional search tree. This means that at this point we have a tree with 14 branches (alternatives). The actual number of alternatives, however, will be determined at the end of the verification stage (Step 6).

Next the system compares the "dossier" of each candidate node with the initial requirements of the problem (Figure 6-2) to determine the contribution of each component. The comparison proceeds according to the procedure presented in Section 5-3. According to the procedure, at the end of the updating operation two blocks of information will be



```
* FUNCTIONAL REPRESENTATION PANEL *

[Elevator, N_flag = 1, E_flag = 1] [search_tree, candidate_nodes]

Alternative 1 (F-can):
    Components:                 [electric_motor (110362, SP)]
    Functions, accomplished:    ["supply_mechanical_energy(rot. mot.)" (2 hp, 3450 rpm)]
    Functions, remaining:       ["adjust_rotational_speed" (2 hp, 3450 rpm, rpm2),
                                "convert_rot_motion_to_lin_motion" (2hp,<rpm2>,<V1>),
                                "adjust_speed" (2 hp, V1, 0.67 ft/sec),
                                And
                                "sense_position" (linear, fixed),
                                "switch_off", "stop_motion (2 hp, <3450 rpm>,
                                <0.67ft/sec>),
                                And
                                "measure_acceleration" (linear, fixed, 32.2 ft/sec^2),
                                "switch_off", "grab" (1150 lb, 32.2 ft/sec^2)]

Alternative 2 (F-can):
    Components:                 [electric_motor (120045, SP)]
    Functions, accomplished:    ["supply_mechanical_energy(rot. mot.)" (2 hp, 1740 rpm)]
```

Figure 6-5: First level of the functional search tree - updated nodes



```
Alternative 6 (F-can):
    Components:                  [electric_gear_motor (2ADBL2663, SP)]
    Functions, accomplished:     ["supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm),
                                 "adjust_rotational_speed"(2 hp, 1750 rpm, 188 rpm)]
    Functions, remaining:        ["convert_rot_motion_to_lin_motion" (2 hp, 188 rpm, V1),
                                 "adjust_speed" (2 hp, V1, 0.67 ft/sec),
                                 And
                                 "sense_position" (linear, fixed),
                                 "switch_off", "stop_motion (2 hp, <1750 rpm>,
                                 <0.67ft/sec>),
                                 And
                                 "measure_acceleration" (linear, fixed, 32.2 ft/sec^2),
                                 "switch_off", "grab" (1150 lb, 32.2 ft/sec^2)]

Alternative 7 (F-can):
    Components:                  [electric_gear_motor (2ADB21010, SP)]
    Functions, accomplished:     ["supply_mechanical_energy(rot. mot.)" (2 hp, 1750 rpm),
                                 "adjust_rotational_speed"(2 hp, 1750 rpm, 125 rpm)]
    Functions, remaining:        ["convert_rot_motion_to_lin_motion" (2 hp, 125 rpm, V1),
                                 "adjust_speed" (2 hp, V1, 0.67 ft/sec),
```

Figure 6-5 (continued)

associated with each node in the search tree. The first block will contain information on what the partial design represented by the node can accomplish, and the second block will tell the system what portion of the initial requirements is still "uncovered".

The results of the updating operation are presented in Figure 6-5 on the system's functional representation panel. To avoid unnecessary listings, only two out of several screens have been illustrated (showing alternatives 1 and 6 completely and alternatives 2 and 7 partially). The description of each alternative comprises a list of its constituent components, an "accomplished functions" section and a "remaining functions" section. Note that :

- The system has bound as many free function specifiers as possible with the available information to this point. For example, in Alternative 6, the free variable "rpm1" (representing the rotational speed of the primary source of motion) has been bound to 1750 rpm, or the original speed of the gear-motor (2ADBL2663);

- Wherever necessary, the system has modified the specifiers of its elemental functions according to its choices of components. In the first line of the "accomplished functions" part of Alternative 6, for example, it has removed the specifier "V1" referring to the speed of the linear motion that could have been supplied by the primary source of motion. Similarly, it has removed the specifier pair <V1, V2> referring to a linear-speed adjustment in case a linear motion had been provided. Both decisions have been based on the fact that the selected component (electric motor) generates rotational motion rather than linear motion;
- To this point, none of the candidate components has exhibited an "extra" (i.e. unintended - see Section 5-3 for definition) function. All the functions offered by the selected components are among those initially requested in the problem.

6.5.4 VERIFYING THE CANDIDATE ALTERNATIVES (STEP 6)

Each of the candidate alternatives is now verified against the initial requirements(Figure 6-2). The verification serves two main purposes: to make sure that each alternative satisfies the external constraints explicitly expressed as part of the initial requirements, and to check the precedence order of its functions against that of the initial requirements.

In this particular problem, no applicable external constraint was included in the requirements[8]. As for the precedence order of the elemental functions, it could be readily determined that all candidates are in compliance with the desired order. Single-function alternatives (numbers 1 to 3 and number 14) are automatically approved. Alternative 4 is approved because in the initial requirements (Figure 6-2) no precedence order was specified between functions 1 (supply_mechanical_energy) and 7 (stop_motion), meaning that they can take place in any relative order including the one offered by the brake motor. Alternatives 5 to 13 are found acceptable because their precedence order is the same as initially requested (adjust_speed *follows* supply_mechanical_energy).

At this point the surviving alternatives (in this case all of them) are stored in a TURBO PROLOG internal database reserved for the functional representation panel of the blackboard.

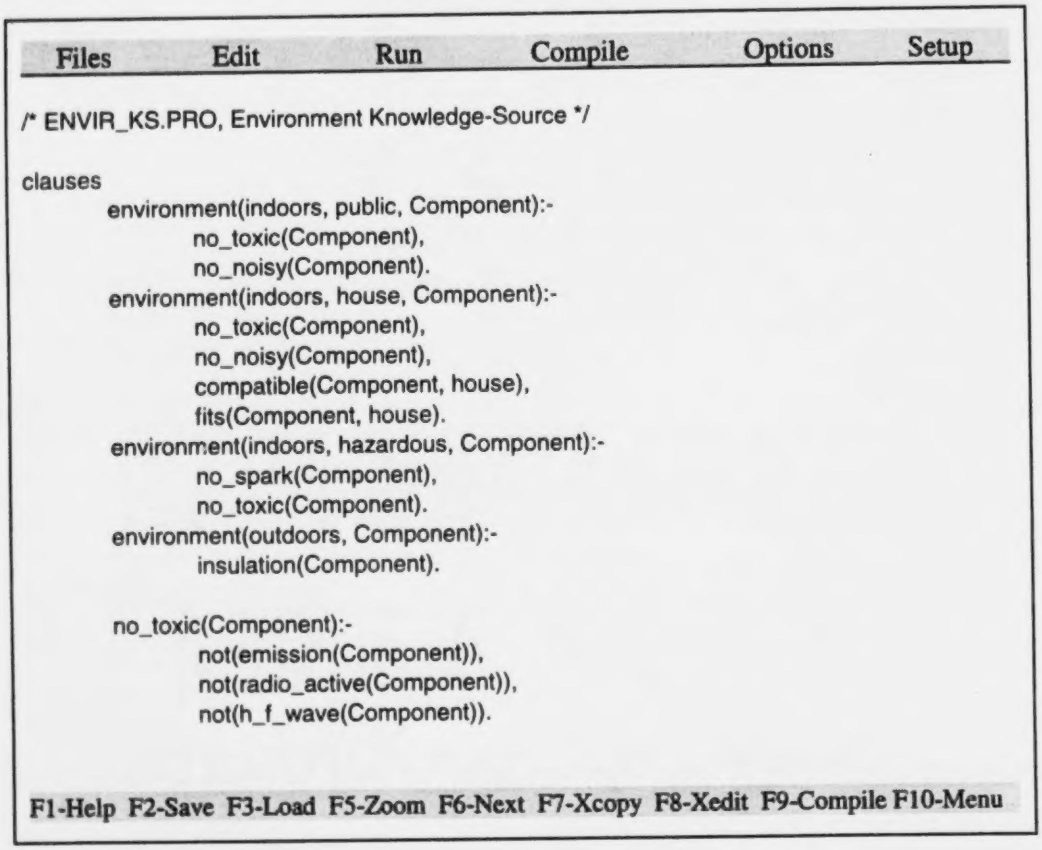
If there were no other constraints (to urge the system to further examine the results), this would terminate one iteration of the design process and would allow all 14 alternatives to proceed to the next iteration. However, the "environment" constraint is there to make sure that each approved design would meet the conditions implied by the user's choice of an "existing house" as the environment wherein the product will be used. As we shall see shortly, this will reject such unjustifiable ideas as using a 3-phase motor in an existing house which presumably is not equipped with a 3-phase power source, or using an i. c. engine in the house where its noise and emissions can cause serious problems for the residents[9].

---

[8] As mentioned in earlier chapters, an *external constraint* is one which is expressed in terms of the problem's design / performance parameters so that they can be processed by the computational methods discussed in Chapter 5. Qualitative constraints (such as the one presented in this example) will be handled by the *critic knowledge-sources*, as will be discussed shortly.

6.6 CRITICIZING THE PARTIAL DESIGNS (STEP 7)

As discussed in Chapter 4, Conceptual Designer's problem-solving paradigm allows various expert agents (knowledge sources) to monitor the evolution of the partial design(s)

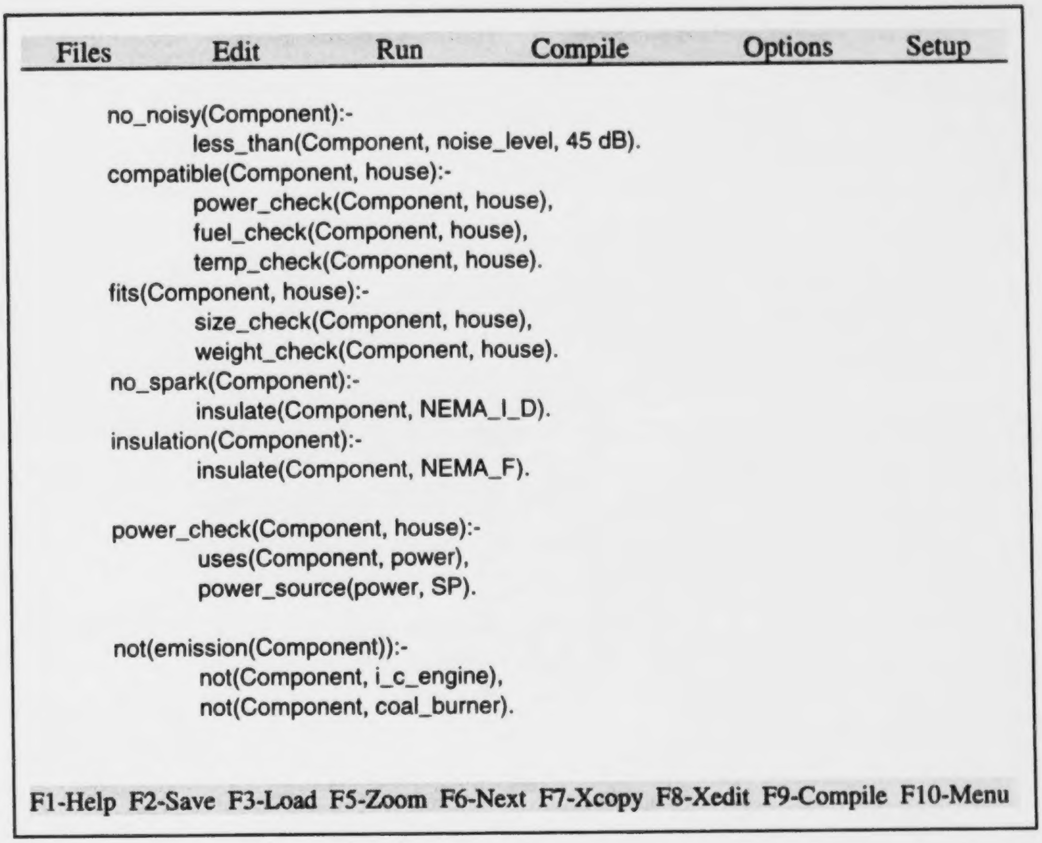

Files
Edit
Run
Compile
Options
Setup
F1-Help F2-Save F3-Load F5-Zoom F6-Next F7-Xcopy F8-Xedit F9-Compile F10-Menu

```
/* ENVIR_KS.PRO, Environment Knowledge-Source */

clauses
        environment(indoors, public, Component):-
                no_toxic(Component),
                no_noisy(Component).
        environment(indoors, house, Component):-
                no_toxic(Component),
                no_noisy(Component),
                compatible(Component, house),
                fits(Component, house).
        environment(indoors, hazardous, Component):-
                no_spark(Component),
                no_toxic(Component).
        environment(outdoors, Component):-
                insulation(Component).

        no_toxic(Component):-
                not(emission(Component)),
                not(radio_active(Component)),
                not(h_f_wave(Component)).
```

Figure 6-6: The *environment* knowledge source

---

[9] The conditions stated here, as well as the contents of the environment knowledge-source presented in the next section, have been postulated for demonstration purposes only, and may or may not be complete or strong enough reasons for rejecting the alternatives in reality.



```
no_noisy(Component):-
        less_than(Component, noise_level, 45 dB).
compatible(Component, house):-
        power_check(Component, house),
        fuel_check(Component, house),
        temp_check(Component, house).
fits(Component, house):-
        size_check(Component, house),
        weight_check(Component, house).
no_spark(Component):-
        insulate(Component, NEMA_I_D).
insulation(Component):-
        insulate(Component, NEMA_F).

power_check(Component, house):-
        uses(Component, power),
        power_source(power, SP).

not(emission(Component)):-
        not(Component, i_c_engine),
        not(Component, coal_burner).
```

Figure 6-6 (continued)

and to evaluate / modify them according to the agents' expertise. For demonstration purposes, in this problem we have considered one such agent, namely the *environment* knowledge source.

Once the verification stage is over, the system looks for signals to invoke knowledge-sources. In this case, it comes across the global constraint *environment,* which has been appointed as the activation signal for the corresponding KS.

The contents of the environment KS are partially shown in Figure 6-6. Written as a set of Prolog's "clauses"[10], it checks the design alternatives component by component against a number of predefined conditions dictated by various working environments.

---

[10] The clauses here have been simplified from their standard Turbo Prolog format for the benefit of those less familiar with the language.

Depending on the environment stated in the problem, the KS fires the proper conditions. Satisfaction of respective conditions by all components in a (partial) design will result in the approval of the design. In this case the KS will give the system the go-ahead to proceed to other knowledge sources (if their activation criteria are met) or to go on to the next stage.

As explained earlier, knowledge sources are mainly composed of sets of if-then statements. The environment KS (Figure 6-6), for example, contains four basic statements each pertaining to a different working environment including public indoor places (e.g. subway stations), private indoor places (e.g. houses), indoor places where hazardous materials are present (e.g. paint shops) and outdoors.

In Prolog, an if-then statement follows the following general format.

```
Result ("then" statement) :-
        Condition ("if" statement) 1,
        Condition ("if" statement) 2,
        ..............
        Condition ("if" statement) n.
```

Meaning that for the "result" to be true, conditions 1 to *n* must be all true. For example, the constraint *"environment (indoors, house, Component)"*, in which the variable "Component" will sequentially be bound to different components of a design, will be satisfied if all four of its conditions "no_toxic(Component))" and "no_noisy(Component))" are satisfied. Later in the program, each of the two conditions is in turn expressed as a "result" statement with its own conditions to be satisfied. This chain of refinements goes on until all sub-conditions of the original condition are covered.

Arguments of the if-then statements can be the attributes of the components or simply the names of some components. In Figure 6-6, for example, one of the sub-conditions, namely the "not(emission(Component))" sub-condition, boils down to naming those components in the components-library that actually emit toxic fumes (e.g. i. c. engines).

In the current example, two of the selected components failed to meet the implications of the environment constraint. One was the three-phase electromotor (alternative 3 in Figure 6-4) which violated the condition "power_check" in Figure 6-6, and the other was the i. c. engine (alternative 14) which did not satisfy the condition "not(emission)". The remaining alternatives were approved because they conformed to the norms of the knowledge source on their pronounced attributes.

As prescribed by step 7, the failed alternatives are removed from the search tree and the remaining are re-numbered.

6.7 UPDATING THE (STRUCTURAL) SEARCH TREE (STEP 8)

The 12 surviving alternatives are mapped onto the structural representation panel (Figure 6-7) where they make the first level of a tree representing the feasible partial designs (physical- rather than functional descriptions).

Since none of the above alternatives is yet covering all the functional requirements stated in the problem, there are no complete designs to report to the user, and hence the system heads for the next iteration (step 9).

Before we start the next iteration at step 2, there are a number of points one should note about what we have covered so far in this chapter.

a. As it is the case with any other computer system, Conceptual Designer heavily relies in its functioning on the information it receives from the user, either as part of the problem statement or in the form of domain knowledge contained in its knowledge sources and databases. The system is NOT intrinsicly aware of the facts which a human designer would take for granted. For example, it will not automatically reject the idea of using a nuclear reactor as the source of energy for a household appliance if the reactor is included in its components-database and if its use for such purposes is not explicitly banned by one of the knowledge sources. The more detailed information the system is provided with, the more accurate its results will be.

   The need for transferring a "sea of information" to the computer should not be considered a drawback for an automated design system, for this is a one-time investment that, in the light of the computers' fast processing and massive data handling capabilities, will pay off very quickly.

   This is especially true with complex problems. The "elevator" example presented here is a small problem which could be solved, perhaps more efficiently, by a human designer and hence does not demonstrate the system's full potentials. One also has to take into account the versatility of the system to independently handle problems from various areas of expertise and its ability to generate multiple solutions simultaneously.

b. As an example for the above argument, we note that so far the system has not come up with any "system constraints" defined in Chapter 3. The reason is that in order to keep the example as simple and brief as possible, we did not include the structural details of the components in their respective knowledge-cells. If, for instance, we had included information such as the output-shaft diameter of each electromotor and its keyway dimensions in the corresponding .OBJ file, Conceptual Designer would have introduced such constraints as "hub_in_dia ≥ mot_shaft_dia" and would have made its satisfaction a condition for the approval of the adjacent components.

c. Critic knowledge sources may or may not be present in the system. In this chapter we included a simple KS to demonstrate how the common engineering knowledge and/or the expertise of domain experts can be encapsulated and used to help guide the design process. The system would nonetheless have proceeded and resulted in multiple solutions even in the absence of the KS. However, it would have been up to the user then to screen the final solutions and to keep the valid ones.

d. Conceptual Designer would consider *all* approved instances of a component-type and would treat each one as an independent alternative. As discussed in earlier chapters, this will give the user more choices and will guarantee that the system will not lose the overall optimum design(s) due to a best-first strategy.

In Figure 6-7 the letter "S" indicates that the tree nodes represent Structural designs rather than functional ones, and the dash within the brackets tells the system that the corresponding design is not complete yet.

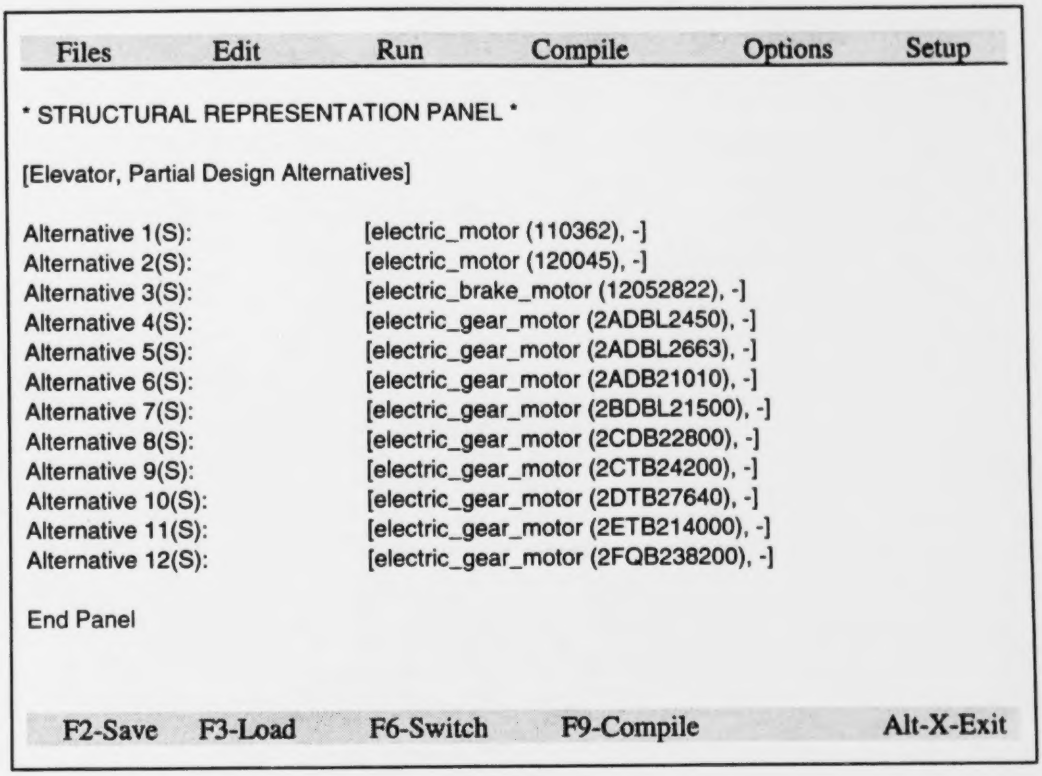
6.8 ITERATING THE DESIGN PROCESS (STEP 9)

Having 12 partial design alternatives on the structural representation panel of its blackboard and a functional dossier for each alternative on its functional representation panel, the system now recursively examines each alternative. This means that it repeats the same steps as taken so far, only this time the initial requirements of the problem are replaced with those listed in each alternative's dossier under "remaining functions". In other words, the original problem is replaced with 12 sub-problems, each having its own requirements.

Except for the exploration step which we promised we would discuss in detail, we will not elaborate the other steps again for the subproblems, and will rather present the results only. The exploration step will be elaborated for alternative 1 (Figure 6-7) whose corresponding functional node was illustrated in Figure 6-5[11]. Unless stated otherwise, the design relations and data used in the rest of this chapter are taken from references (Gieck and Gieck 1990; Baumeister 1978; Juvinall and Marshek 1983; PARKER Catalog).

---

[11] Note that the alternatives have been re-numbered since Figure 6-5. For example, alternative 5 in Figure 6-7, which will be examined shortly for the exploration step, was number 6 in Figure 6-5.



```
* STRUCTURAL REPRESENTATION PANEL *

[Elevator, Partial Design Alternatives]

Alternative 1(S):          [electric_motor (110362), -]
Alternative 2(S):          [electric_motor (120045), -]
Alternative 3(S):          [electric_brake_motor (12052822), -]
Alternative 4(S):          [electric_gear_motor (2ADBL2450), -]
Alternative 5(S):          [electric_gear_motor (2ADBL2663), -]
Alternative 6(S):          [electric_gear_motor (2ADB21010), -]
Alternative 7(S):          [electric_gear_motor (2BDBL21500), -]
Alternative 8(S):          [electric_gear_motor (2CDB22800), -]
Alternative 9(S):          [electric_gear_motor (2CTB24200), -]
Alternative 10(S):         [electric_gear_motor (2DTB27640), -]
Alternative 11(S):         [electric_gear_motor (2ETB214000), -]
Alternative 12(S):         [electric_gear_motor (2FQB238200), -]

End Panel
```

Figure 6-7: Feasible partial design alternatives at the first level of the search tree

LEVEL 1, ALTERNATIVE 1 (Figure 6-7)

The next elemental function to consider is "convert_rot_motion_to_lin_motion" or as we shall briefly call it "linearize_motion" (the reason for this choice was explained earlier in this chapter). Browsing through the database index (Appendix D) Conceptual Designer finds 6 components/ subsystems capable of performing the specified function, viz.

- A cable-hoist (wire-rope being wound around a cylindrical drum),
- A chain-hoist (chain being wound around a cylindrical drum),
- A hydraulic drive (pump and hydraulic cylinder),
- A pneumatic drive (compressor and pneumatic cylinder),
- A rack and pinion drive,
- A power-screw drive

The slider-crank mechanism is not selected, despite its capability to convert rotational motion to linear motion, because it generates *reciprocal* linear motion which is not wanted.

Starting with the cable-hoist, the system opens the corresponding component-cell (cabl_hst.obj) (Figure 6-8). In order to form the "knowledge pool" of the component, it then adds the initial specifications of the sub-problem to the cell. These specifications include the output speed of the electromotor (3450 rpm) which is assigned to variable "CHDRPM" (the rotational speed of the hoist's drum) and the desired linear speed of 0.67 ft/sec (40.2 ft/min) which is assigned to the speed of the cable (variable "CHRS").

The system then activates the exploration KS to solve the given constraint set. In this case, the actual run of the program expectedly resulted in a failure message, implying that the direct connection of a cable hoist to a 3450 rpm electromotor was not a practical idea In terms of the constraints, the failure stemmed from the violation of an inequality constraint in the component's knowledge pool (Figure 6-8).

```
/* "cabl_hst.obj", Turbo Prolog 2.0 */

/* Definition of Terms
CHDD:          drum diameter (inches)
CHDRPM:        drum rotational speed (rpm)
CHDRP:         drum radial pressure (psi)
CHDW:          drum width (inches)
CHRD:          rope diameter (inches)
CHRS:          rope speed (ft/min)
CHRT:          rope tention (lb)
CHRAS:         allowable tensile stress of rope (psi) */

/* Equality Constraints */
.............................................
CHDRPM = 3450
CHRS = 40.2

/* Inequality Constraints */
.............................................
CHDD ≤ 40 × CHRD
CHRS ≤ 1000
```

Figure 6-8: Knowledge-pool of the cable-hoist (partially shown)

The problem arose when the system chose a 0.5" diameter wire rope and calculated the outside diameter of a drum to be connected to the electromotor shaft and to wind the rope at a speed of 40.2 ft/min. The calculation yielded a diameter of 0.04" for the drum which obviously violated the constraint "$40 \times CHRD \leq CHDD$" which requires that the drum diameter be at least 40 times the diameter of the rope (Oberg and Jones 1971).

As a result of this failure, Conceptual Designer issues a *system constraint* to declare the acceptable range of the one function specifier that has caused the failure, namely the input speed to the cable-hoist. We just argued that the speed should provide for a drum diameter of ($40 \times 0.5" = 20"$) or more. This translates to a drum speed of 7.7 rpm or less ($\text{CHDRPM} \leq 7.7$).

As prescribed by the "analysis order" included in the initial presentation of the problem, the system then goes back and considers elemental function 2, (adjust_rotational_speed) while binding the function's first specifier (transmitted power) to 2 hp and its second specifier (input speed) to 3450 rpm and constraining its third specifier (output speed) to 7.7 rpm.

Scanning the database index again, the system finds 6 options for adjusting the rotational speed of the electromotor, viz.

- A spur-gear drive
- A helical-gear drive
- A bevel-gear drive
- A worm-gear drive
- A belt drive
- A chain drive

The next step would be to explore each of these alternatives[12]. Starting with the spur-gear drive, the exploration module opens the corresponding component-cell (spr_gear.obj) and adds the initial specifications to it. The resulting constraint set is shown in Figure 6-9. The exploration module is then activated to solve the constraint set. As in the case of the cable hoist, in the actual run, Conceptual Designer issued a failure message for the obvious reason that reducing the input speed of 3450 rpm to a desired speed of 7.7 rpm or less requires a gear ratio of 448:1 or more, and this is far beyond the maximum recommended gear ratio of 8:1 for a spur-gear set (see inequality constraints in Figure 6-9).

This failure, however, should not result in totally ignoring the component, for one could always break up the required overall gear ratio and carry it out in a number of steps. For this reason, certain component-cells are equipped with a *refinement procedure* that

---

[12] To avoid confusion, one must carfully keep track of the alternatives being generated. For each of the 6 options presented here, the system is likely to find multiple feasible instances. Each of these instances will then be considered in conjunction with each of the 6 "motion linearizers" presented earlier in this section, and the process will continue in this way.

```
/* "spr_gear.obj", Turbo Prolog 2.0 */

/* Definition of Terms
SGSR:           Speed Ratio
SGIS:           Input Speed
SGOS:           Output Speed
SGTP:           Transmitted Power
SGCD:           Center Distance
SGFD:           Face Width
SGNPT:          Number of Pinion Teeth
SGNGT:          Number of Gear Teeth
SGML:           Module
SGAT:           Applied Torque
SGTL:           Transmitted Load
SGMBS:          Maximum Bending Stres
SGALS:          Allowable Stress
SGY:            Lewis Form Factor
SGKV:           Velocity Factor */
................................................

/* Equality Constraints */
SGAT = SGTP / SGIS
SGCD = SGML * SGNPT * (1 + SGSR) / 2
SGTL = SGTP * 60 / SGML / SGNPT / SGIS / PAI
SGOS = SGIS / SGSR
SGSR = SGNGT / SGNPT
SGMBS = SGTL / SGKV / SGFW / SGML / SGY
SGKV = 360 / (SGML * SGNPT * SGIS * PAI + 360)
SGY = -0.000000018 * (^(SGNPT, 4)) + 0.0000046 * (^(SGNPT, 3))
        - 0.00045 * (^(SGNPT, 2)) + 0.02 * SGNPT + 0.05
SGIS = 3450
SGTP = 1.47

/* Inequality Constraints */
SGMBS ≤ SGALS
SGSR ≤ 8.0
9 * SGML ≤ SGFW
SGFW ≤ 16 * SGML
SGOS ≤ 7.7
```

Figure 6-9: The constraint set of the spur-gear drive

would allow them to get around this impass[13]. According to the procedure, the violated constraint would be rendered active (i.e. the ≤ sign would be replaced with the equal sign) and the conflicting constraint(s) would be removed. The resulting, consistent constraint set would then be solved to give the specifications of the component instance which will carry out the desired function partially. In other words, Conceptual Designer would find a component instance with function specifiers as close to the desired values as permitted by the constraints.

The initial requirements of the problem would then be modified to reflect the "remaining" portion of the elemental function in question. This would consist of replacing the original values of the specifiers with their new values as well as including a modified version of the then-conflicting constraint to the component-cell.

In the case of the spur gear drive, the procedure would set the gear ratio to 8 (the maximum allowed by the constraints) instead of the desired value of 448, and would remove the constraint "SGOS ≤ 7.7" from the constraint set. This refined constraint set would now be solved by the exploration module to give a (number of) gear set(s) with a gear ratio of 8:1. Execution of the program resulted in the 4 alternatives presented in Figure 6-10.

SGSR	SGNGT	SGNPT	SGML	SGOS
8.0	112	14	3.0	431 rpm
8.0	120	15	2.5	431 rpm
8.0	96	12	3.0	431 rpm
8.0	136	17	2.0	431 rpm

Figure 6-10: Multiple solutions to the spur-gear sub-problem

Meanwhile, the system modifies the description of the "remaining" functions for this node on the functional representation panel of the blackboard. This is done by changing the values of the specifier "input speed" of function 2 (Figure 6-2) from 3450 rpm to 431 rpm. Alternatively stated, the system is now asked to find other component(s) to finish the partially-performed task of reducing the initial speed of 3450 rpm to a final speed of 7.7 rpm or less.

---

[13] As a rule of thumb, the procedure applies to those components whose function specifiers are "adjustable". Definition of an adjustable specifier was given in Chapter 5.

Another search is conducted with the above purpose and the system finds the same six candidates it found before for accomplishing the speed-adjustment task. These candidates are the spur-, helical-, bevel- and worm-gear drives plus the belt drive and the chain drive. For each of these candidates the system goes through the very same steps as it did for the spur-gear drive and faces the same problem (the required reduction ratio of 56:1 or more is beyond the speed-reducing ability of a single component).

As in case of the spur-gear drive, the system then sets each component's reduction ratio to its maximum and finds its feasible instances. Figure 6-11 briefly presents the results of this round of instantiations. The reduction ratio shown in the figure for each component is the maximum recommended value for that component (Oberg and Jones 1971; Quayle 1985; Shigley and Mischke 1989).

Component	Reduction Ratio	No. of Instances Found	Output Speed
spur_gear_drive	8:1	5	54 rpm
helical_gear_drive	12:1	3	36 rpm
bevel_gear_drive	10:1	3	43 rpm
worm_gear_drive	40:1	1	11 rpm
belt_drive	10:1	7	43 rpm
chain_drive	6:1	4	72 rpm

Figure 6-11: Alternative ways to further reduce the speed of the electromotor's shaft

Normally, the next step would be to modify the requirements of the problem and perform another round of instantiations on the same six components listed in Figure 6-11. The goal would then be to find component instances that would yet further reduce the output speeds presented in the figure to 7.7 rpm or less.

At this point one may notice the rate at which the search tree (Figure 6-12) is growing. Thus far we have considered only the first alternative (out of 12 presented in Figure 6-7) for performing function 1. Then for this single alternative the system has offered six ways to carry out the first "instalment" of the second function and, only for one of these six, yet another six ways to carry out the second instalment. Considering that even the second function is not yet fully carried out, one could realize the combinatorial explosion in progress.

In previous chapters we discussed a "pruning" mechanism to avoid this situation. In the present example we have adopted a two-tiered pruning criterion with two sets of

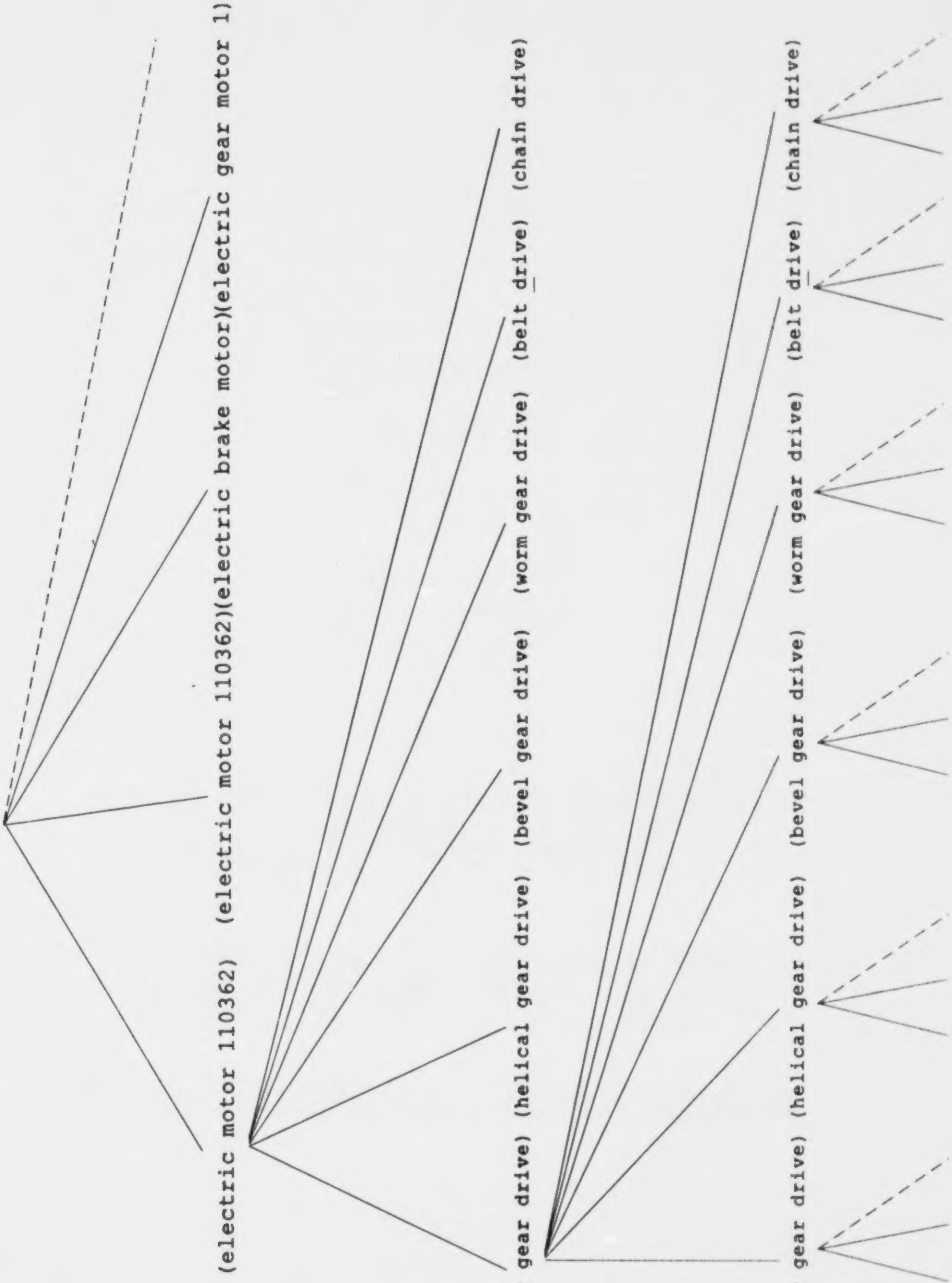

(electric motor 110362)
(electric motor 110362)(electric brake motor)(electric gear motor 1)
(spur gear drive)
(helical gear drive)
(bevel gear drive)
(worm gear drive)
(belt drive)
(chain drive)
(spur gear drive)
(helical gear drive)
(bevel gear drive)
(worm gear drive)
(belt drive)
(chain drive)

Figure 6-12: Partial search tree

activation conditions. The first tier would be applied to all alternatives in the same level of the search tree. It would remove from the tree any design alternative in which a single function requires more than two[14] components/subsystems to be carried out, unless there is no other choice. In other words, Conceptual Designer is told to count the number of "installments" to which a single elemental function has to be divided before it could be carried out completely. Should the function not be completed after two consequtive corresponding partial design would be discarded.

This "sub-criterion" is always active regardless of the number of alternatives in a tree level. It is meant to rid the system from less-efficient design ideas which tend to achieve the desired tasks the longer, usually more expensive way.

The second tier of the adopted pruning criterion would be activated once the number of design alternatives in a single level of the search tree exceeded 20. Once activated, it would look for alternatives with similar structures, i.e. the ones that comprise the same components-types (and not necessarily component-instances). If there were more than one such alternative, the system would keep the first one and discard the rest. This would also apply to the alternatives with some functions mapped to more than one component if at least one of these components is shared by a number of alternatives.

For example, if the system comes up with the two solutions

- [electromotor, 3500 rpm] [helical gear set, ratio 12:1] [.....]
- [electromotor, 1750 rpm] [helical gear set, ratio 6:1] [.....]

and there are more than 20 alternatives in the last level of the search tree, then it would keep the first alternative and discard the second, as the two solutions consist of the same component-types, though different component-instances. Also, if the solutions differ only in *some* of the component-types resulting from mapping of a single function, such as

- [electromotor, 3500 rpm] [helical gear set, ratio 6:1] [spur gear set, ratio 2:1] [.....]
- [electromotor, 1750 rpm] [helical gear set, ratio 3:1] [bevel gear set, ratio 2:1] [.....]

then the system would again discard the second alternative, as the two solutions share one of the component-types (namely the helical gear set) in the mapping of the function "adjust speed".

The rationale behind this pruning sub-criterion is that one implementation (instance) is enough to represent a design idea. Alternatively stated, whenever a pruning is necessary, the system should keep those alternatives that represent different design ideas, rather than various instances of the same (or almost the same) idea. Although in general this pruning scheme (and any scheme for that matter) is in contrast with our principle of "saving all alternatives to insure the best solution", in this particular example we accept the risk of losing that best solution because we are only concerned about design *ideas* and not design *instances,* and because we are not given sufficient information to evaluate various designs quantitatively and thus to choose the best one.

---

[14] This number could be changed by the user, if the problem requires and CPU resources were available.

Returning to the example we notice that according to Figure 6-11, no component-instance has so far been found capable of bringing the speed down to 7.7 rpm, a requirement dictated by the cable hoist. Therefore, in accordance with the first tiere of the above criterion, all partial designs found so far (i.e. the ones with speed adjusters) are discredited and discarded.

Having expanded the first node of the second level of the search tree of Figure 6-12 (i.e. the spur-gear drive), the system now backtracks and considers the next node (the helical-gear drive) for expansion. As before, the given specifications are added to the respective component-cell (hel_gear.obj) to form the knowledge pool of the component. Again, these specifications include "in. speed = 3450 rpm" and "transmitted power = 2 hp"plus the constraint "out. speed ≤ 7.7 rpm".

The same scenario is repeated: Conceptual Designer spots a violated constraint ("gear ratio ≤ 12" contradicted by the required ratio of 448:1), the refinement procedure is activated, the ratio is set to 12:1 and the inconsistent constraint "output speed = 3450 rpm" is removed from the constraint set, the new constraint set is solved and in this case 2 feasible instances are found. The instantiation process is then repeated to find the "mating" component(s) (to complete the unfinished task of reducing the speed to less than 7.7 rpm).

Recalling the maximum reduction ratios of various "speed adjusters" from Figure 6-12, and considering that the system's pruning criterion limits the number of components in each configuration to two, it readily follows that from the four possible configurations (helical-spur, helical-helical, helical-bevel and helical-worm-gear) with maximum combined reduction ratios of (12 × 8, 12 × 12, 12 × 10 and 12 × 40 respectively), only the last configuration survives the pruning procedure as it is potentially capable of bringing the speed down to 7.7 rpm or less. Conceptual Designer was able to find two feasible instances of the helical-gear drive and one feasible instance of the worm-gear drive for each one[15]. The resulting two solutions to the sub-problem are presented in Figure 6-13.

---

[15] Note that due to the non-deterministic nature of a GA search, neither the number nor the identity of the instances of a component found through the search is necessarily the same in each iteration. Whereas a previous run of the GA routine had resulted in, for example, 3 instances of helical-gear drives (Figure 6-11), this time it found only 2 instances.

	Helical Gear/Pinion	Worm/Gear	Output Speed
Alt. 1:	144 teeth / 12 teeth (ratio 12:1)	1 thread / 38 teeth (ratio 38:1)	7.6 rpm
Alt. 2:	168 teeth / 14 teeth (ratio 12:1)	1 thread / 40 teeth (ratio 40:1)	7.2 rpm

Figure 6-13: The two solutions to the speed-adjustment sub-problem

The next node in the search tree to expand is the bevel-gear drive. An attempt by the system to instantiate this component resulted in another failure message, as one of the constraints in the component's constraint set was violated by the input speed of 3450 rpm. The violated constraint was the one that limits the pitch-line velocity of a bevel gear to 1000 ft/min (Shigley and Mischke 1989). To satisfy the constraint, a pinion with a pitch diameter of one inch or less would have to be mounted on the output shaft of the electromotor; and this would be inconsistent with the strength requirements of the gear. This option was therefore discarded and the system considered the next option, namely a worm-gear drive.

As for the worm-gear drive, the system expectedly chose the highest reduction ratio allowed (40:1) and tried to augment it with a second speed-reducer. Of all the options available to it (Figure 6-11), the system found only two components capable of performing the desired task, a helical-gear set and a second worm-gear set. The other options were eliminated as they had maximum reduction ratios of 10:1 or less, which could not bring the speed down to the desired level. The resulting solutions for this sub-problem are illustrated in Figure 6-14.

	First Component	Second Component	Output Speed
Alt. 1:	worm-gear (40 teeth/1 thread)	helical-gear set (161 teeth/14teeth)	7.5rpm
Alt. 2:	worm-gear (40 teeth/1 thread)	helical-gear set (144 teeth/12 teeth)	7.2 rpm
Alt. 3:	worm-gear (40 teeth/1 thread)	worm-gear (24 teeth/2 thread)	7.2 rpm

Figure 6-14: Results of the expansion of the worm-gear node

Figures 6-13 and 6-14 present the five solutions to the sub-problem (i.e. to carry out function 2 "adjust rotational speed" for the cable-hoist). These are all the solutions the system could find, as the other potential candidates (belt drive and chain drive) cannot reduce the speed to 7.7 rpm or less in one or two steps.

Following the "analysis order" of the problem (Figure 6-2), the system now augments the five partial solutions presented above with the component "cable-hoist" it had picked earlier. For each of the five alternatives, the output speed of the sub-system is unified with the input (drum) speed of the cable-hoist and the resulting knowledge-pool is explored by Conceptual Designer.

If an instance of the cable-hoist is found which can wind the wire rope at a speed of 4.2 ft/min as a result of the drum speed specified by one of the five alternatives above then the system will no longer need to consider functions 4 and 5 in Figure 6-2 (i.e. "adjust_linear_speed" and "redirect_linear_motion"), as the rope can be directly connected to the elevator box and lift it at the desired speed. Otherwise, the system will have to find additional components to carry out those functions.

In the current example, since Conceptual Designer has already examined function 3 with the final desired speed of 40.2 ft/min assigned to its output speed and has back-propagated the implications of this assignment to the previous function (function 2), one can expect that the system will readily find feasible instance(s) of the hoist and hence will skip functions 4 and 5.

The actual instantiation of the cable-hoist led to the very same results. Feasible instances of the hoist were found for the five speed-adjusters presented above and hence five partial solutions (speed-adjuster plus cable-hoist) were reported (Figure 6-15). All five solutions complied with the requirements of the global (environment) constraint of the problem. They also satisfied the functional precedence order prescribed by the problem. Therefore they were all marked as feasible partial designs and were reported to the structural representation panel of the blackboard.

Having fully expanded the "cable-hoist" node of the search tree, the system back-tracks to the parent node and finds five more child-nodes to be expanded. These nodes represent the alternative ways to convert rotational motion to linear motion. They include the "chain-hoist", the "hydraulic-drive", the "pneumatic-drive", the "rack-and-pinion-drive" and the "power-screw-drive".

Using a similar approach to exploring these nodes as the one used in the case of the cable-hoist, Conceptual Designer considers those nodes one by one and reports the results to the blackboard. To avoid repetitious discussions, we shall skip the details of the exploration process for the five nodes and will rather present the results. The following is a brief diary of the process including the more important results.

Alternative 1: [helical_gear_drive (12/144 teeth, 30 deg_helix)] [worm_gear_drive (1 thread/38 teeth, 20 deg_lead)] [cable_hoist (8×19-0.5 in_rope, 20.3 in_dia_drum]
Alternative 2: [helical_gear_drive (14/168 teeth, 15 deg_helix)] [worm_gear_drive (1 thread/40 teeth, 15 deg_lead)] [cable_hoist (8×21-0.5 in_rope, 21.4 in_dia_drum]
Alternative 3: [worm_gear_drive (1thread/40 teeth, 25 deg_lead)] [helical_gear_drive (161/14 teeth, 15 deg_helix)] [cable_hoist (6×19-0.5 in_rope, 20.5 in_dia_drum]
Alternative 4: [worm_gear_drive (1thread/40 teeth, 20 deg_lead)] [helical_gear_drive (144/12 teeth, 15 deg_helix)] [cable_hoist (6×21-0.56 in_rope, 22.4 in_dia_drum]
Alternative 5: [worm_gear_drive (1thread/40 teeth, 15 deg_lead)] [worm_gear_drive (2 thread/24 teeth, 25 deg_lead)] [cable_hoist (8×25-0.44 in_rope, 21.3 in_dia_drum]

Figure 6-15: Solutions to the speed-adjuster/cable-hoist sub-problem

- Examining the chain hoist, Conceptual Designer faced the violation of the speed constraint and backed up to function 2 where it again found three ways to reduce the speed of the electromotor. It then found matching hoist-instances for each speed reducer and reported the following three solutions (Figure 6-16).

Alternative 1: [helical_gear_drive (14/140 teeth, 30 deg_helix)] [worm_gear_drive (1 thread/35 teeth, 20 deg_lead)] [chain_hoist (0.5×1.34 in_chain, 15.6 in_dia_drum]
Alternative 2: [worm_gear_drive (1thread/38 teeth, 20 deg_lead)] [helical_gear_drive (144/12 teeth, 15 deg_helix)] [chain_hoist (0.625×1.875 in_rope, 20.3 in_dia_drum)]
Alternative 3: [worm_gear_drive (2thread/42 teeth, 25 deg_lead)] [worm_gear_drive (1 thread/18 teeth, 15 deg_lead)] [chain_hoist (0.5×1.34 in_chain, 16.8 in_dia_drum]

Figure 6-16: Solutions to the speed-adjuster/chain-hoist sub-problem

- Next the hydraulic-drive option was explored. This time, the system did not come across a violated speed-constraint, as the initial port of the sub-system, namely the rotor of the pump, *can* be directly connected to the electromotor shaft. The desired final speed of 40.2 ft/min as well as an additional speed equal to half of that value were sequentially assigned to the output[16], linear speed of the sub-system (the piston speed) and the speed of the electromotor shaft was assigned to the input speed of the pump. The resulting knowledge-pool was then explored and the four solutions presented in Figure 6-17 were reported.

Alternative 1: [gear_pump (3450 rpm, 125 psi, 24 gpm)] [hydraulic_cylinder (4.4 in_bore, 2.18 in_rod, 40.2 ft/min, 16 ft_stroke)]
Alternative 2: [gear_pump (3450 rpm, 200 psi, 16.4 gpm)] [hydraulic_cylinder (5.0 in_bore, 2.25 in_rod, 20.1 ft/min, 8 ft_stroke)]
Alternative 3: [vane_pump (3450 rpm, 75 psi, 46 gpm)] [hydraulic_cylinder (6.0 in_bore, 2.8 in_rod, 40.2 ft/min, 16 ft_stroke)]
Alternative 4: [vane_pump (3450 rpm, 175 psi, 14 gpm)] [hydraulic_cylinder (5.0 in_bore, 2.8 in_rod, 20.1 ft/min, 8 ft_stroke)]

Figure 6-17: Solutions to the hydraulic-drive sub-problem

The results presented in Figure 6-17 shows that the system has found two feasible instances of the hydraulic drive that can move the elevator box 16 feet at a speed of 40.2 ft/min. It has also found two instances of the same mechanism that can raise the box 8 feet at a speed of 20.1 ft/min. If another mechanism could be found to double the displacement of the box (and hence its speed, as the time is constant), then one would end up getting the same outcome while saving the cylinder half of its stroke. This option, though maybe economically infeasible, was included here to demonstrate the possibility of hinting the system to explore non-trivial options.

- The fourth option, i.e. the pneumatic drive, was next explored. As in the case of hydraulic cylinder, the system unify the initial (electromotor) and final (elevator) specified speeds with the input and output speeds of the candidate component and was able to find two feasible instances of it (Figure 6-18).
- In the case of rack and pinion, two options are considered in the component cell: a fixed pinion with a moving rack which would lift the elevator box , and a fixed rack with a "climbing" pinion attached to the box. The difference being, among other things, that in the former case the rack would be subject to buckling whereas in the latter it would not.

---

[16] As an example of how the DbE model allows for the exploitation of domain-specific knowledge, we have included in the component-cells of hydraulic- and pneumatic-drives the optional choice of half or full piston stroke with the same stroke time. In the half-stroke case, a displacement magnifier (doubler) mechanism would be then added to the system to provide the desired full displacement and, since the stroketime remains the same, full speed. The cells are set up so that the exploration module would consider both options and result in two (sets of) solutions.

Alternative 1: [vane_pump (3450 rpm, 80 psi, 34.4 gpm)] [pneumatic_cylinder (5.0 in_bore, 2.0 in_rod, 40.2 ft/min, 16 ft_stroke)]

Alternative 2: [vane_pump (3450 rpm, 100 psi, 26.2 gpm)] [pneumatic_cylinder (6.0 in_bore, 2.0 in_rod, 20.1 ft/min, 8 ft_stroke)]

Figure 6-18: Solutions to the pneumatic-drive sub-problem

In both cases Conceptual Designer confronted constraint violations due to the high speed of the electromotor, and issued system constraints to restrain the rotational speed of the pinion between 9.6 rpm and 35 rpm (due to upper and lower limits on the pitch diameter of the pinion and its hub diameter). It then sought speed reduction prior to instantiating the rack and pinion mechanism. The results are illustrated in Figure 6-19.

Alternative 1: [helical_gear_drive (14/84 teeth, 20 deg_helix)] [worm_gear_drive (1 thread/25 teeth, 15 deg_lead)] [rack_f.pinion_drive (16.0 ft_rack, 306 teeth_rack, 0.63 in_c.pitch, 34 teeth_pinion, 5 d.pitch]

Alternative 2: [worm_gear_drive (1 thread/38 teeth, 25 deg_lead)] [spur_gear_drive (15/105 teeth, 5 d.pitch)] [rack_f.pinion_drive (16.0 ft_rack, 123 teeth_rack, 1.57 in_c.pitch, 25 teeth_pinion, 2 d.pitch]

Alternative 3: [bevel_gear_drive (16/64 teeth, 14/76 deg_pitch)] [worm_gear_drive (1 thread/34 teeth, 20 deg_lead)] [rack_f.pinion_drive (16.0 ft_rack, 367 teeth_rack, 0.52 in_c.pitch, 38 teeth_pinion, 6 d.pitch]

Alternative 4: [bevel_gear_drive (14/140 teeth, 6/84 deg_pitch)] [helical_gear_drive (14/154 teeth, 15 deg_helix)] [f.rack_pinion_drive (16.0 ft_rack, 192 teeth_rack, 1.0 in_c.pitch, 21 teeth_pinion, 3 d.pitch]

Alternative 5: [belt_drive (A64, 2/18 in_p.dia)] [helical_gear_drive (12/144 teeth, 30 deg_helix)] [f.rack_pinion_drive (16.0 ft_rack, 306 teeth_rack, 0.63 in_c.pitch, 23 teeth_pinion, 5 d.pitch]

Alternative 6: [worm_gear_drive (1thread/35 teeth, 20 deg_lead)] [belt_drive (B66, 3/18 in_p.dia)] [f.rack_pinion_drive (16.0 ft_rack, 245 teeth_rack, 0.78 in_c.pitch, 40 teeth_pinion, 4 d.pitch]

Alternative 7: [worm_gear_drive (1thread/36 teeth, 15 deg_lead)] [chain_drive (80, 67 in_chain, 1 in_pitch, 15/60 teeth_sprocket)] [f.rack_pinion_drive (16.0 ft_rack, 184 teeth_rack, 1.0 in_c.pitch, 20 teeth_pinion, 3 d.pitch]

Figure 6-19: Results of the expansion of the rack-and-pinion node

In the description of the components in Figure 6-19, the term "f.rack_pinion_drive" denotes a fixed rack with "climbing" pinion whereas the term "rack_f.pinion_drive" refers to a fixed pinion with a moving rack. Also the terms "c.pitch" and "d.pitch" refer to "circular pitch" and "diametral pitch" respectively.

- The same procedure was followed in the case of the power-screw-drive. Again the system ended up reducing the speed of the electromotor before instantiating the candidate drive. In this case, however, only one option (fixed screw with climbing nut) was considered. The results of the exploration stage are presented in Figure 6-20.

Alternative 1: [spur_gear_drive (20/96 teeth, 10 d.pitch)] [power_screw_drive (2.25-0.33p-0.67L-Acme)]
Alternative 2: [helical_gear_drive (15/108 teeth, 15 deg_helix)] [power_screw_drive (2.5-0.33p-1.00L-Acme)]
Alternative 3: [worm_gear_drive (2thread/15 teeth, 20 deg_lead)] [power_screw_drive (2.5-0.33p-1.00L-Acme)]
Alternative 4: [bevel_gear_drive (25/120 teeth, 12/78 deg_pitch)] [power_screw_drive (2.25-0.33p-0.67L-Acme)]
Alternative 5: [belt_drive (A50, 2.5/12 in_p.dia)] [power_screw_drive (2.75-0.33p-0.67L-Acme)]
Alternative 6: [chain_drive (60, 65 in_chain, 0.75 in_pitch, 10/72 teeth_sprocket)] [power_screw_drive (2.5-0.33p-1.00L-Acme)]

Figure 6-20: Results of the expansion of the power-screw node

- This completed the expansion of the first leaf-node of the search tree (Figure 6-7) at this level. This means that the system had explored all possibilities (according to its database) of carrying out, partially or completely[17], functions 1 through 4 (supply mechanical power in the form of motion and convert the motion to a linear one with a desired speed) using a 2 hp, 3450 rpm electromotor.

  The expansion of this leaf-node resulted in 27 partial design alternatives. This met the second tiere of the pruning criterion and activated the mechanism. The redundant alternatives were removed from the tree and that reduced the number of remaining designs to 13 (Figure 6-21).

---

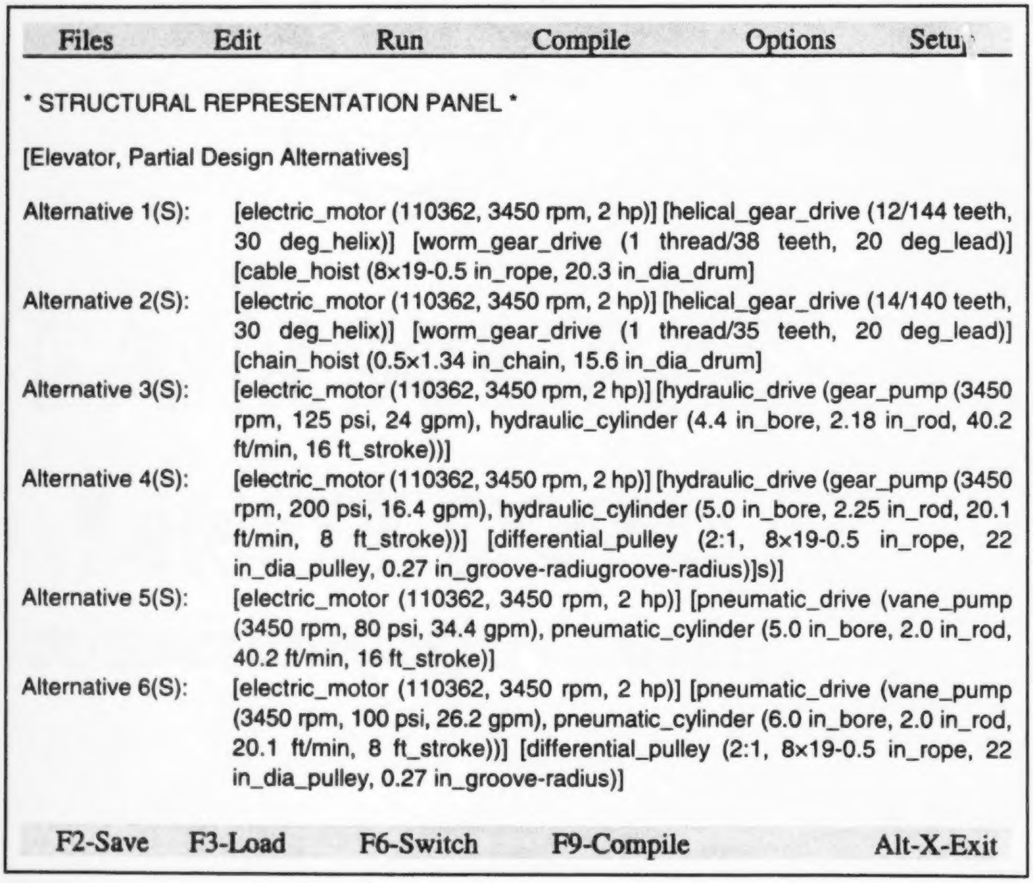
[17] The term *partial* refers to those designs that would provide for half the desired speed, namely the ones using a type 2 hydraulic/pneumatic drive to linearize the rotational motion as discussed earlier. The next step in augmenting these designs would be to find components that can double their linear speed/displacement. We shall further explain this shortly.

LEVEL 1, ALTERNATIVES 2, 3 AND 4 (Figure 6-7)

Conceptual Designer then expanded the other nodes at this level before going to the next level and further augmenting the partial designs in Figure 6-21. The results of these expansions will be presented shortly in this chapter. The details of the expansions, however, will not be discussed.

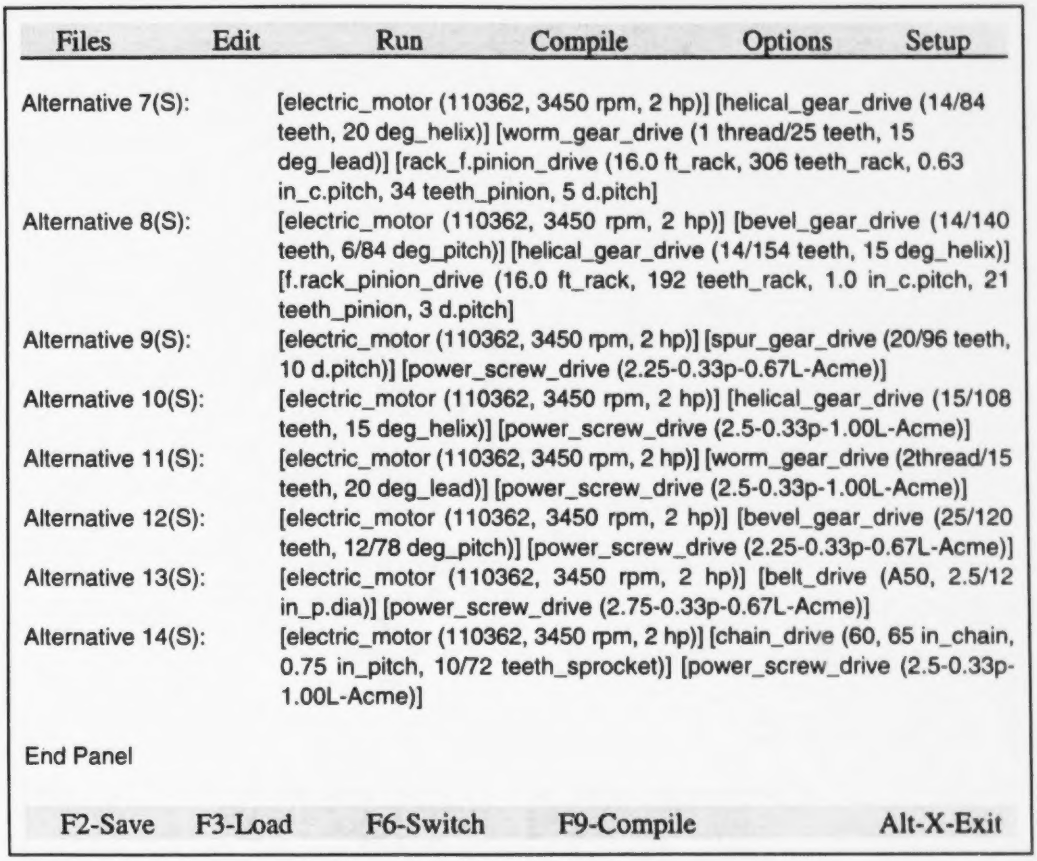
LEVEL 2, ALL ALTERNATIVES

The first leaf-node to be expanded at the next level of the search tree was the (electromotor-helical gear drive-worm gear drive-cable hoist) (Figure 6-15). The first of the remaining functions for this node was to "sense position". This function is necessary for determining when the elevator box must be stopped.



```
* STRUCTURAL REPRESENTATION PANEL *

[Elevator, Partial Design Alternatives]

Alternative 1(S):   [electric_motor (110362, 3450 rpm, 2 hp)] [helical_gear_drive (12/144 teeth,
                    30 deg_helix)] [worm_gear_drive (1 thread/38 teeth, 20 deg_lead)]
                    [cable_hoist (8×19-0.5 in_rope, 20.3 in_dia_drum]
Alternative 2(S):   [electric_motor (110362, 3450 rpm, 2 hp)] [helical_gear_drive (14/140 teeth,
                    30 deg_helix)] [worm_gear_drive (1 thread/35 teeth, 20 deg_lead)]
                    [chain_hoist (0.5×1.34 in_chain, 15.6 in_dia_drum]
Alternative 3(S):   [electric_motor (110362, 3450 rpm, 2 hp)] [hydraulic_drive (gear_pump (3450
                    rpm, 125 psi, 24 gpm), hydraulic_cylinder (4.4 in_bore, 2.18 in_rod, 40.2
                    ft/min, 16 ft_stroke))]
Alternative 4(S):   [electric_motor (110362, 3450 rpm, 2 hp)] [hydraulic_drive (gear_pump (3450
                    rpm, 200 psi, 16.4 gpm), hydraulic_cylinder (5.0 in_bore, 2.25 in_rod, 20.1
                    ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8×19-0.5 in_rope, 22
                    in_dia_pulley, 0.27 in_groove-radiugroove-radius)]s)]
Alternative 5(S):   [electric_motor (110362, 3450 rpm, 2 hp)] [pneumatic_drive (vane_pump
                    (3450 rpm, 80 psi, 34.4 gpm), pneumatic_cylinder (5.0 in_bore, 2.0 in_rod,
                    40.2 ft/min, 16 ft_stroke)]
Alternative 6(S):   [electric_motor (110362, 3450 rpm, 2 hp)] [pneumatic_drive (vane_pump
                    (3450 rpm, 100 psi, 26.2 gpm), pneumatic_cylinder (6.0 in_bore, 2.0 in_rod,
                    20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8×19-0.5 in_rope, 22
                    in_dia_pulley, 0.27 in_groove-radius)]
```

Figure 6-21: Pruned search tree after expanding the first leaf-node



```
Alternative 7(S):     [electric_motor (110362, 3450 rpm, 2 hp)] [helical_gear_drive (14/84
                      teeth, 20 deg_helix)] [worm_gear_drive (1 thread/25 teeth, 15
                      deg_lead)] [rack_f.pinion_drive (16.0 ft_rack, 306 teeth_rack, 0.63
                      in_c.pitch, 34 teeth_pinion, 5 d.pitch]
Alternative 8(S):     [electric_motor (110362, 3450 rpm, 2 hp)] [bevel_gear_drive (14/140
                      teeth, 6/84 deg_pitch)] [helical_gear_drive (14/154 teeth, 15 deg_helix)]
                      [f.rack_pinion_drive (16.0 ft_rack, 192 teeth_rack, 1.0 in_c.pitch, 21
                      teeth_pinion, 3 d.pitch]
Alternative 9(S):     [electric_motor (110362, 3450 rpm, 2 hp)] [spur_gear_drive (20/96 teeth,
                      10 d.pitch)] [power_screw_drive (2.25-0.33p-0.67L-Acme)]
Alternative 10(S):    [electric_motor (110362, 3450 rpm, 2 hp)] [helical_gear_drive (15/108
                      teeth, 15 deg_helix)] [power_screw_drive (2.5-0.33p-1.00L-Acme)]
Alternative 11(S):    [electric_motor (110362, 3450 rpm, 2 hp)] [worm_gear_drive (2thread/15
                      teeth, 20 deg_lead)] [power_screw_drive (2.5-0.33p-1.00L-Acme)]
Alternative 12(S):    [electric_motor (110362, 3450 rpm, 2 hp)] [bevel_gear_drive (25/120
                      teeth, 12/78 deg_pitch)] [power_screw_drive (2.25-0.33p-0.67L-Acme)]
Alternative 13(S):    [electric_motor (110362, 3450 rpm, 2 hp)] [belt_drive (A50, 2.5/12
                      in_p.dia)] [power_screw_drive (2.75-0.33p-0.67L-Acme)]
Alternative 14(S):    [electric_motor (110362, 3450 rpm, 2 hp)] [chain_drive (60, 65 in_chain,
                      0.75 in_pitch, 10/72 teeth_sprocket)] [power_screw_drive (2.5-0.33p-
                      1.00L-Acme)]

End Panel
```

Figure 6-21 (continued)

In its components-database, Conceptual Designer found only one component capable of performing this functions, namely the "proximity_switch (923AA2XM_A7T_L)" (MICRO SWITCH Catalog). The component was added to the above node and (later on) to all nodes for which the function "sense position" was being explored.

Exploration of the proximity-switch cell revealed that, in addition to the task for which it was selected (senseing the position), it could perform the function "switch on/off" by sending a signal to the respective device. Hence functions 5 and 6 were removed from the requirements list of the above-mentioned nodes. For these nodes, the "remaining functions" list now began with "stop_motion (2 hp, <3450 rpm>, <0.67 ft/sec>)".

- For those partial designs with output speeds equal to half of the desired speed (e.g. alternatives 2 and 4 in Figure 6-17 and alternative 2 in Figure 6-18) the expansion of the node resulted in the selection of a differential pulley, as it was the only component in the database capable of adjusting linear motion. A differential pulley with a ratio of 2:1 (i.e. doubling the displacement and hence the speed of the elevator box) was added to these alternatives (Figure 6-22).

- [electric_motor (110362)] [hydraulic_drive (gear_pump (3450 rpm, 200 psi, 16.4 gpm), hydraulic_cylinder (5.0 in_bore, 2.25 in_rod, 20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8×19-0.5 in_rope, 22 in_dia_pulley, 0.27 in_groove-radius)]
- ....
- [electric_motor (110362)] [hydraulic_drive (vane_pump (3450 rpm, 175 psi, 14 gpm), hydraulic_cylinder (5.0 in_bore, 2.8 in_rod, 20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8×19-0.5 in_rope, 22 in_dia_pulley, 0.27 in_groove-radius)]
- ....
- [electric_motor (110362)] [pneumatic_drive (vane_pump (3450 rpm, 100 psi, 26.2 gpm), pneumatic_cylinder (6.0 in_bore, 2.0 in_rod, 20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8×19-0.5 in_rope, 22 in_dia_pulley, 0.27 in_groove-radius)]
- ....

Figure 6-22: Typical partial designs with half-stroke hydraulic/pneumatic drives

Note that at this stage, the partial designs just mentioned were "two functions behind" the others (i.e. the ones already containing a proximity switch). While the latter nodes covered functions 1 through 6, the former group only covered up to function 4. One could therefore expect that not all branches of the search tree would reach the goal stage (performing all required functions) at the same time.

For convenience, we shall now divide the partial designs generated thus far into four groups and, in the rest of this section, shall refer to each group by its number. A partial design may or may not contain a half-stroke hydraulic/pneumatic drive and a differential pulley. Also, it may or may not already contain a "locking" element (e.g. a brake-motor, a worm-gear drive and a power-screw drive). We shall refer to those designs containing a half-stroke drive and no locking elements as "group 1" designs and to those with the same drive and a locking element as "group 2" designs. Designs that contain neither a half-stroke drive nor a locking element are referred to as "group 3" designs and those without a half-stroke drive but with a locking element as "group 4" designs.

LEVEL 3, ALL ALTERNATIVES

- Proceeding to the next level of the tree, the four groups of designs mentioned above were treated as follows.

For "group 1" and "group 2" partial designs the next function was "sense_position". A mapping of this function resulted in the addition of a proximity switch (similar to the one selected before) to the corresponding designs.

For "group 3" partial designs the next function to consider was "stop_motion (2 hp, <3450 rpm>, <0.67 ft/sec>)". Conceptual Designer found one component to carry this out, namely a disk brake [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]. The component was then added to corresponding partial designs.

- As for "group 4" partial designs, the next function was function 8 (measure_acceleration). Function 8 is part of a batch of three functions (functions 8 through 10 in Figure 6-2) meant as a safety precaution against the possible "free drop" of the elevator box. To avoid this worst scenario, the acceleration of the box would be continually monitored and if it equalled that of the gravity, a "grabbing" mechanism would stop the box from falling.

For function 8 (measure acceleration), Conceptual Designer found a single component (accelerometer AX-535-502) (Motion Control Catalog)] with a sensing range of 0.03 g - 20.0 g. As in the case of the proximity switch, the accelerometer could also switch off the electromotor (function 9) by sending it a signal. Therefore, for this group of partial designs, the list of "remaining" functions shrank to one function: "grab the box".

LEVEL 4, ALL ALTERNATIVES

- Proceeding to yet the next level of the search tree, the system mapped the function "stop_motion" for "group 1" designs and augmented these nodes with a disk-brake similar to the one previously added to "group 3" designs.

For "group 2" and "group 3" designs the next function was "measure-acceleration". Again, the same accelerometer (AX-535-502) was added to the partial designs of these two groups.

- The next group of designs to consider was "group 4", for which the system now had to map function "grab" (the elevator box). Conceptual Designer found no components to carry out this function and therefore issued a failure message. No precautions had been made to rid the system from this impass.

This situation underscores one of the main characteristics of the Conceptual Designer: that it *designs through mapping of given functions to the components present in its database*. If such a mapping cannot be performed, either because the function is not a Standard Elemental Function (Chapter 1) or because no component in the database can carry it out, the system will terminate the design process and issue a failure message.

The last group of functions (functions 8 to 10) was intentionally included in the problem to highlight this characteristic of the model. This being done, and our purpose in this chapter being to demonstrate the functioning of the DbE model rather than actually designing a particular machine, we then modified the original problem to exclude function 8 to 10, and undid any steps Conceptual Designer had made to carry out those functions. This implied the removal of the accelerometer from "group 2" and "group 3" designs.

- Faced with the modified problem, Conceptual Designer continued the solution process by checking all 35 partial design alternatives left on the blackboard after undoing the implications of functions 8 to 10. None of the remaining alternatives met the pruning criterion, hence they were all saved on the structural representation panel, although their number exceeded 20.

  The system then examined the functional description of each design and compared it with the original requirements. All designs were found to have satisfied the entire set of functional requirements. They were therefore marked "complete" and presented as final solutions. Each of the 35 final solutions is shown in a separate window in Figure 6-23 at the end of this chapter. For visualization purposes, we have added a simple sketch to each alternative.

Navigating through the steps that took the system from the initial state of the problem to its goal state, one realizes the importance of the assertions that form the foundation of the DbE model. First of all, we notice that the whole process is based on a series of careful mappings of the elemental functions to physical components. The mappings are not necessarily one-to-one. Some functions (e.g. adjust-speed) have been mapped to more than one component and some components (e.g. a gear-motor) contribute to more than a single function.

Exploration of the candidate components before each mapping is central to the functioning of the model. If, for example, the function "linearize-motion" had been directly mapped to the component "cable-hoist" without the constraint set of the component being explored, the system would not have become aware of the conflicting constraints and would have prescribed that the hoist be directly connected to a 3450 rpm electromotor. Or if the system had not studied the functional behavior of the brake-motor to learn about its secondary function "stop-rotational-motion" while choosing it for the function "supply-motion", it would unnecessarily have used an additional brake to carry out the former function.

The other fundamental "axiom" of DbE states that only by considering all (partial) alternatives can one guarantee the system's arival at the optimal solution. A best-first search strategy is likely to be deluded into selecting a sub-optimal or even infeasible path, just because at some point along the path there is a node which dominates its fellow nodes at its level. For example, if at the first level of the search tree the system had chosen the option "electromotor" over the other options (brake-motor and gear-motor) because it costs the least, it could have missed the optimal final solution, possibly among the final designs containing a brake- or gear-motor.

As discussed earlier, a *selection module* (based on a given optimization criterion) can be added to the program to evaluate each final alternative and choose the optimal one(s). This, however, would require detailed information about those aspects of the design on which the optimization criterion is based.

In practice, many of the commonly used criteria (e.g. cost, manufacturability, dimensions and weight) revolve around the structural specifications of the designs. We also note that conceptual design, by definition, is not expected to determine these characteristics, as its job is to decide the major, function-defining components only. For instance, if the desired function is to adjust a rotational speed, conceptual design may suggest a gear-drive and/or a V-belt-drive, but will normally not mention the shafts and bearings required to support them.

This makes the traditional selection (optimization) task virtually impossible in this case. Nonetheless, Conceptual Designer *is* prepared to carry out the optimization process should the user decide to settle for a criterion based only on the major components of a design (such as cost/weight/manufacturability of major components). As an example of this, in the current example we made the program generate a simple "comparison table" based on the number of major components in each final design. The table is shown in Figure 6-24, with "design numbers" referring to those in Figure 6-23.

6-9 COMPARISON OF RESULTS WITH THOSE OF HUMAN DESIGNERS

As mentioned earlier, the elevator design problem considered here was also assigned to two groups of senior undergraduate students of mechanical engineering. In this section we shall briefly present the design ideas generated by these groups, and then make a comparison between those ideas and the ones generated by the conceptual designer while highlighting some of the more important similarities and differences between the two.

Figures 6-25 and 6-26 schematically represent the ideas generated by the two student groups. An asterisk (*) to the left of a schematic caption indicates the alternative of choice by the group.

Design Number	1	2	3	4	5	6	7	8	9	10	11	12	13
Number of Major Components	6	6	5	6	5	6	6	6	4	4	4	4	3

Design Number	14	15	16	17	18	19	20	21	22	23	24
Number of Major Components	4	5	5	4	5	4	5	5	5	4	4

Design Number	25	26	27	28	29	30	31	32	33	34	35
Number of Major Components	4	4	4	4	4	4	5	6	5	4	4

Figure 6-24: Number of major components in each of the final design alternatives

A quick comparison of the ideas presented in these two figures with those presented in Figure 6-23 shows that except for schematics (6-25-f / 6-26-f) and (6-26-b), all the other basic designs by the student groups are also generated by the Conceptual Designer (C.D.).

Regarding the two exceptions, the first one (a scissors mechanism) had *not* been included in the system's components database or otherwise the system would have chosen it for its function "amplify displacement", just as it chose a differential pulley for the same purpose, and therefore would have come up with the very same design as well.

As for the second exception (DC motor and cable pulley), the system *did* initially choose a DC motor, but as we mentioned earlier, this option was later rejected on the basis of enviroment considerations, that is, due to the fact that direct current is normally not available in residential buildings.

The above comparison supports the following proclamations that we have made in this work:

- Based on the DbE model, the C.D. is capable of generating a set of design ideas (conceptual design instances) comparable to, and often richer than, those produced by human designers.
- In doing so, it thoroughly relies on the information embedded in its knowledge- and data-bases. C.D. cannot "improvise" in unprecedented situations, nor can it suggest components that are not contained in its components database. The richer its knowledge sources, the more design concepts it will be able to generate.

- Both "domain-expertise" and "experience" *can*, to a certain extent, be encapsulated and incorporated in automated design systems. In the current example, the student groups benefited from the expertise and experience of elevator designers through consulting certain companies and examining existing elevator configurations. C.D., on the other hand, relied solely on its experts modules both in generating raw design ideas and in evaluating and screening those ideas. If we adopted the popular definition of Artificial Intelligence, i.e. "the study of how to make computers do things at which, at the moment, people are better" (Rich 1991), we could certainly claim that C.D. demonstrates some degree of AI.
- Generally, the information generated by C.D. is not enough for design optimization in a practical sense. That is, one cannot apply common optimality critera such as weight, cost, etc. to design alternatives as presented by the C.D. This is quite natural, as C.D.'s only commitment is to find embodiments (configurations of physical elements) for the *specified functions*, and very often this does not lead to a detailed design wherein the cost, weight and other characteristics of the entire design can be determined and used as a measure of desirability of the design. Consequently, unlike the student groups, Conceptual Designer did not attempt to choose a single design as the optimal one. Had the necessary information been initially specified, or the optimality criteria been based only on the factors that could be measured in the products of C.D., the system could have easily carried out the optimization.

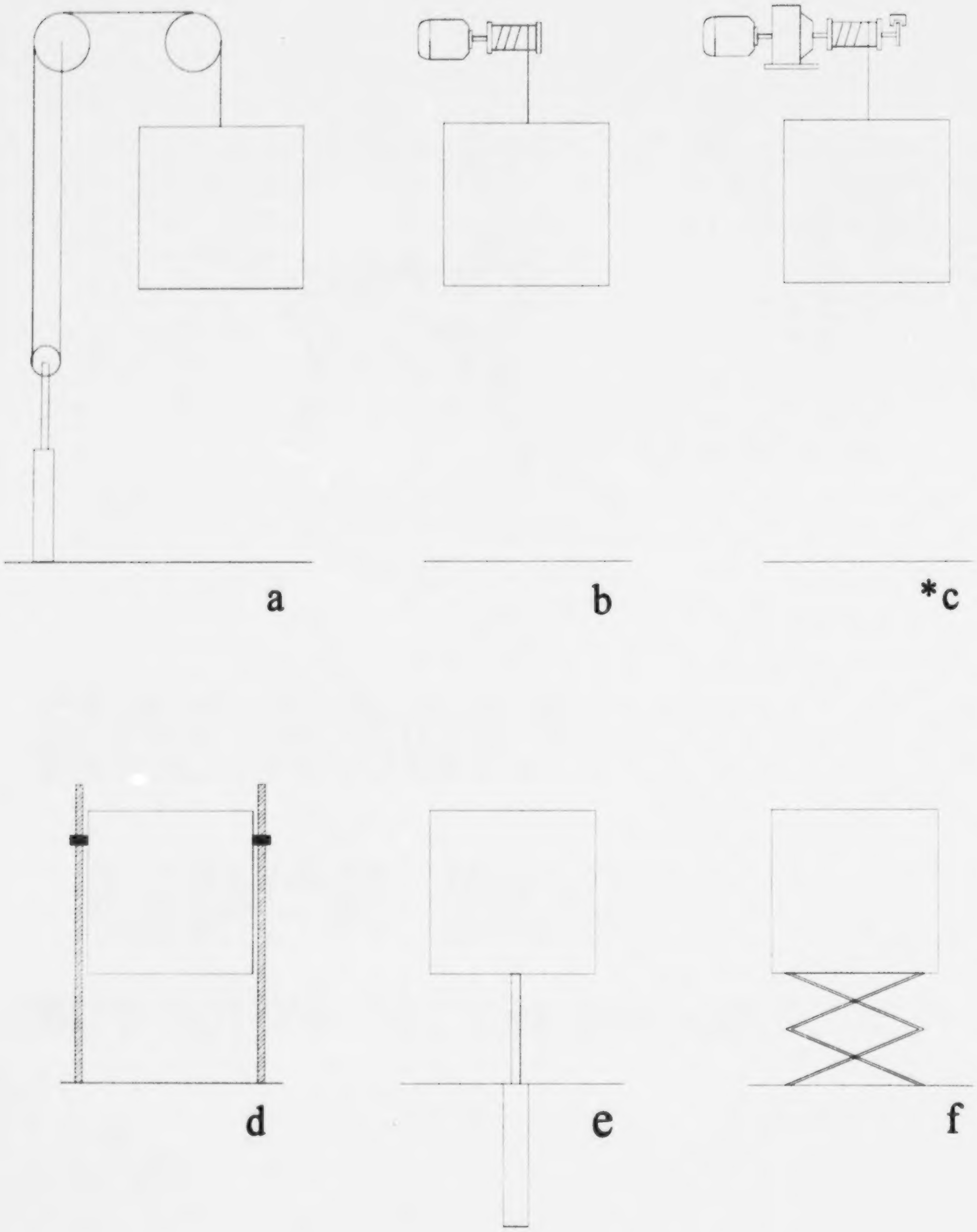


Figure 6-25: Student-group design ideas for the elevator - first group. a) hydraulic cylinder with differential pulley b) cable pulley c) cable pulley-details d) power- screw e) hydraulic cylinder f) scissors mechanism

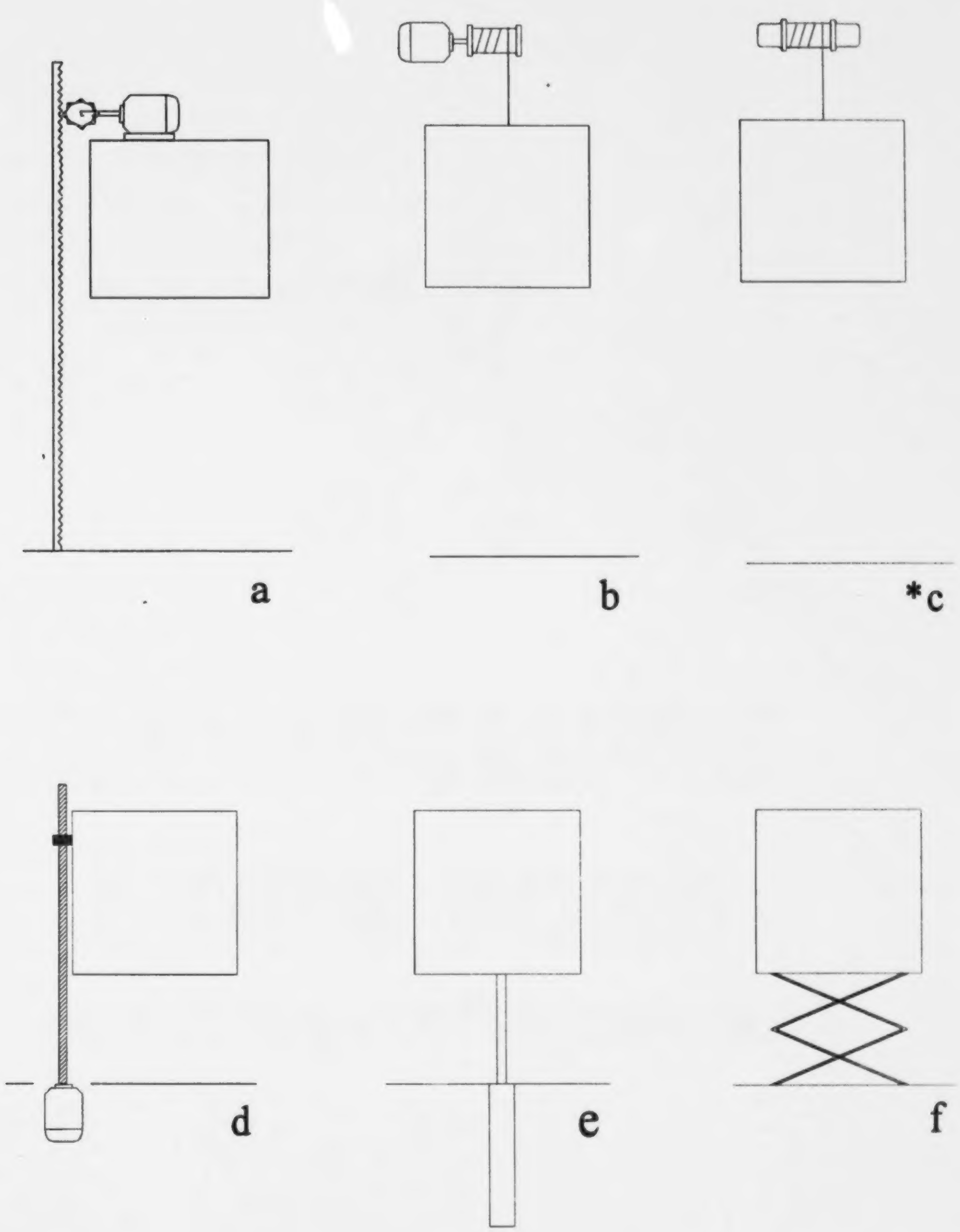
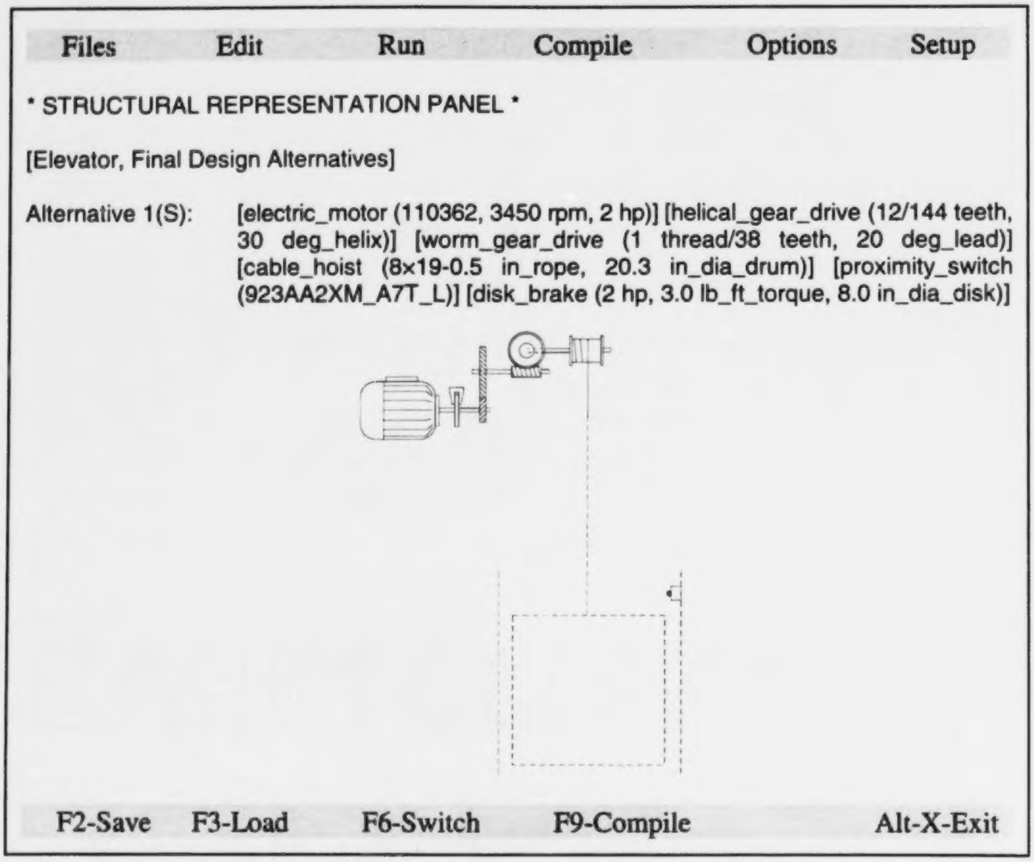


Figure 6-26: Student-group design ideas for the elevator - second group. a) rack and pinion
b) DC motor and cable pulley c) AC winch d) power screw
e) hydraulic cylinder f) scissors mechanism


Files
Edit
Run
Compile
Options
Setup

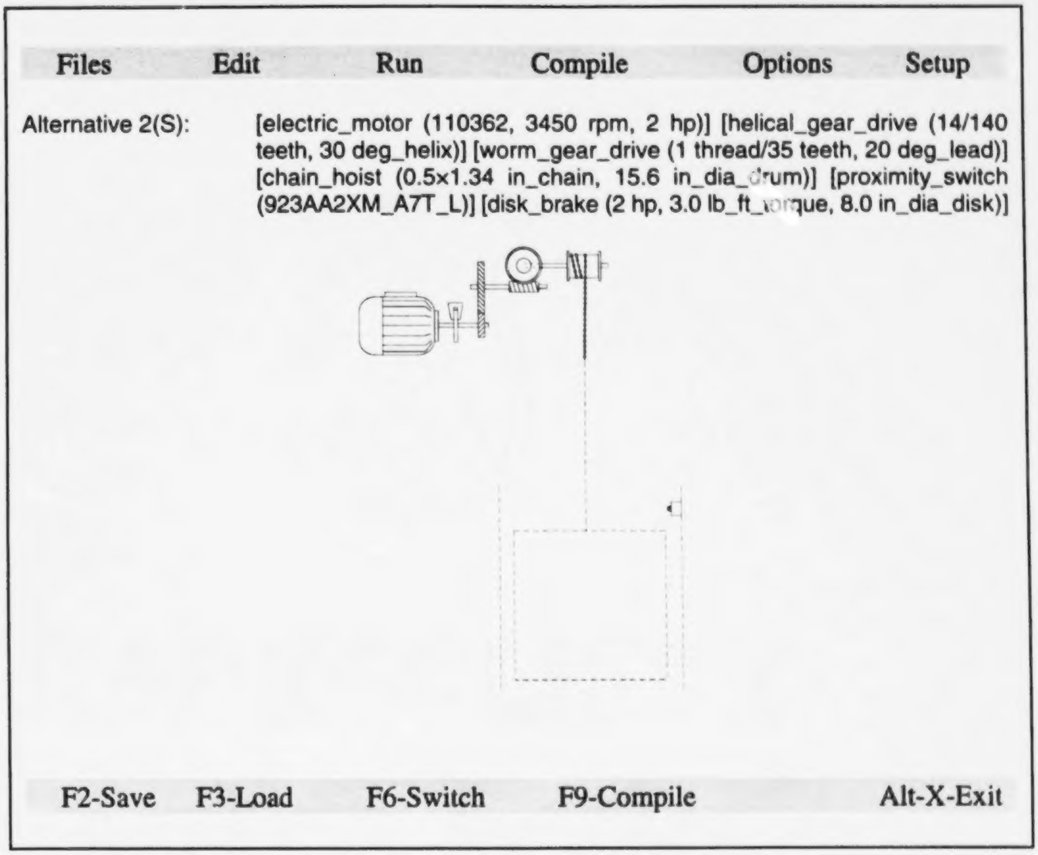
* STRUCTURAL REPRESENTATION PANEL *

[Elevator, Final Design Alternatives]

Alternative 1(S): [electric_motor (110362, 3450 rpm, 2 hp)] [helical_gear_drive (12/144 teeth, 30 deg_helix)] [worm_gear_drive (1 thread/38 teeth, 20 deg_lead)] [cable_hoist (8x19-0.5 in_rope, 20.3 in_dia_drum)] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

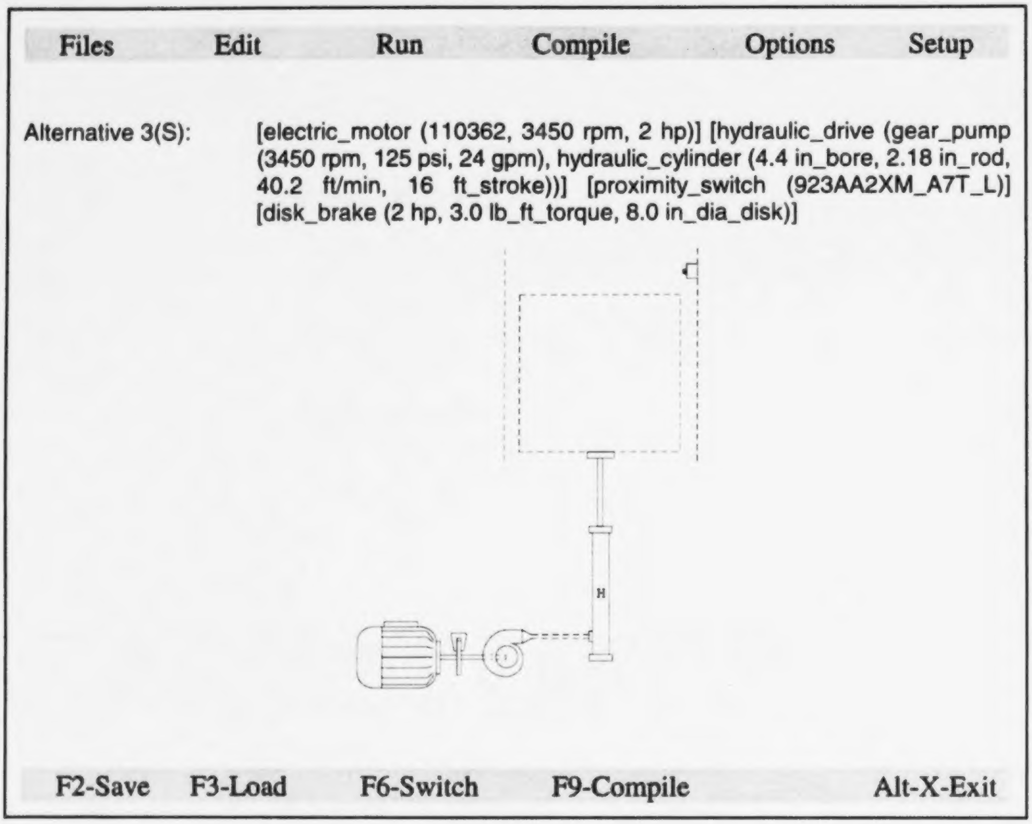
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Figure 6-23: Final design alternatives


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

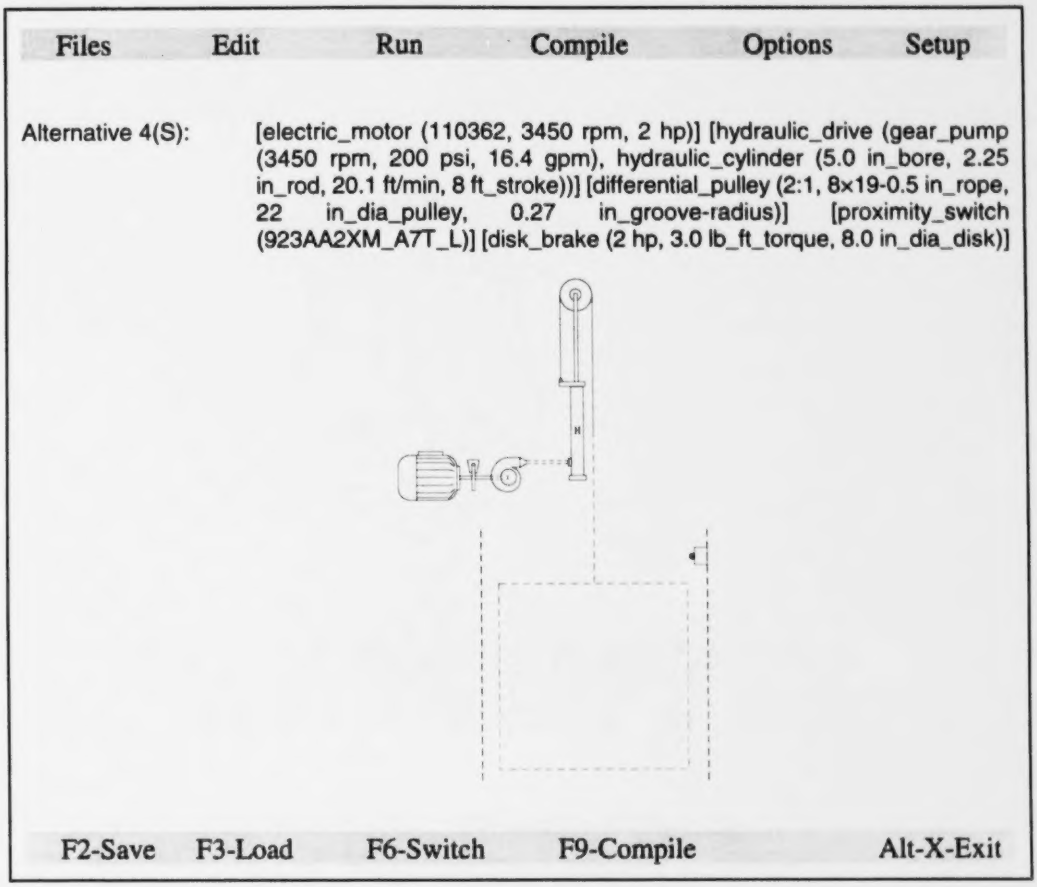
Alternative 2(S): [electric_motor (110362, 3450 rpm, 2 hp)] [helical_gear_drive (14/140 teeth, 30 deg_helix)] [worm_gear_drive (1 thread/35 teeth, 20 deg_lead)] [chain_hoist (0.5×1.34 in_chain, 15.6 in_dia_drum)] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
H
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Alternative 3(S): [electric_motor (110362, 3450 rpm, 2 hp)] [hydraulic_drive (gear_pump (3450 rpm, 125 psi, 24 gpm), hydraulic_cylinder (4.4 in_bore, 2.18 in_rod, 40.2 ft/min, 16 ft_stroke))] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

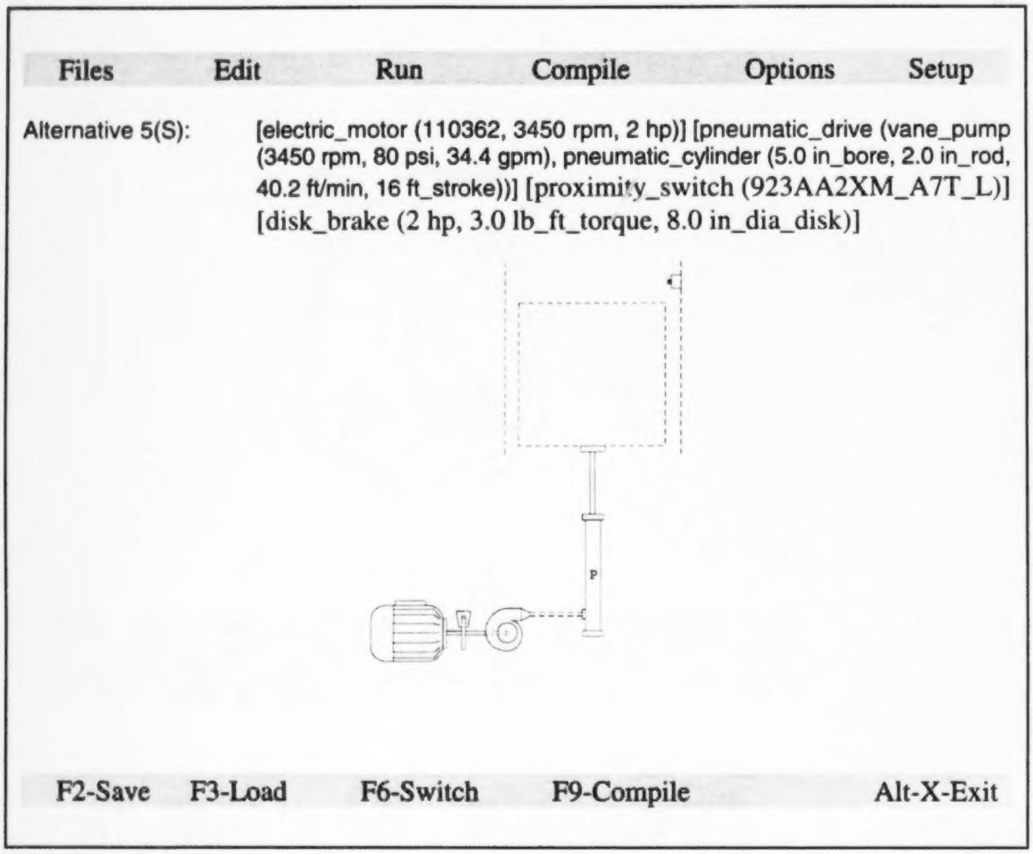
Figure 6-23: Final design alternatives (Continued)



Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit
H

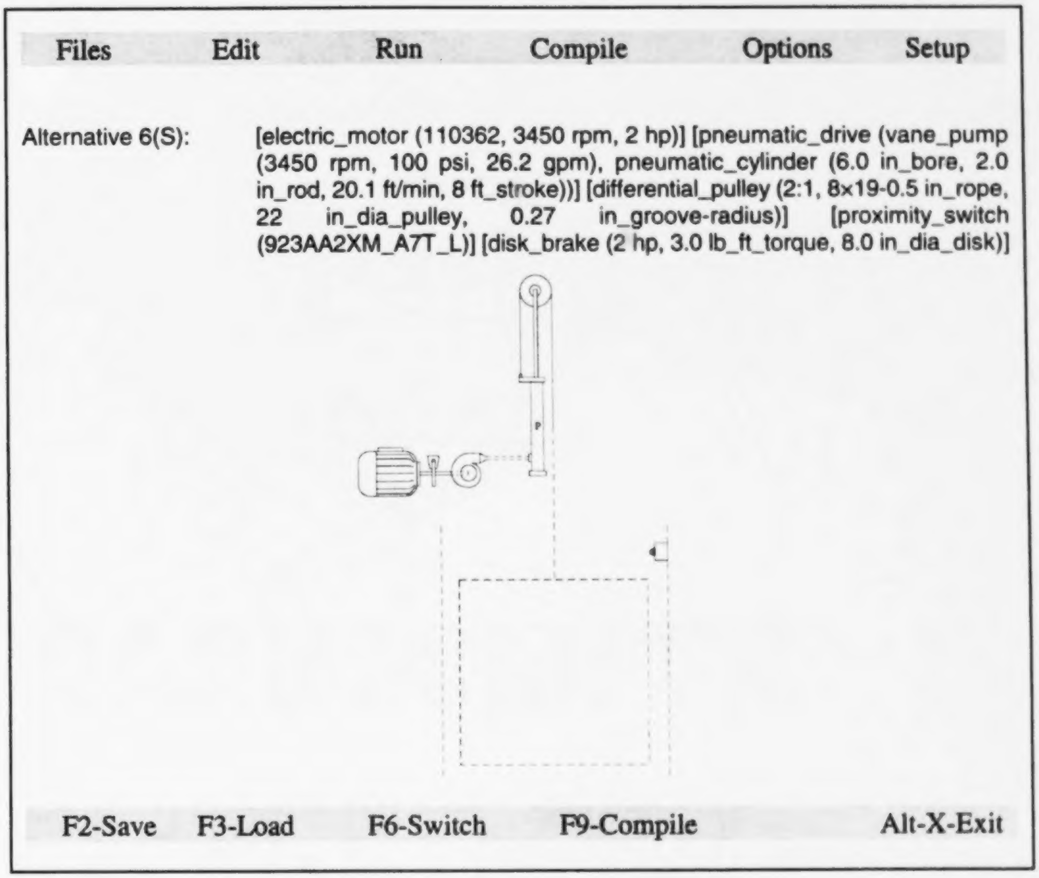
Alternative 4(S): [electric_motor (110362, 3450 rpm, 2 hp)] [hydraulic_drive (gear_pump (3450 rpm, 200 psi, 16.4 gpm), hydraulic_cylinder (5.0 in_bore, 2.25 in_rod, 20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8×19-0.5 in_rope, 22 in_dia_pulley, 0.27 in_groove-radius)] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
P
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Alternative 5(S): [electric_motor (110362, 3450 rpm, 2 hp)] [pneumatic_drive (vane_pump (3450 rpm, 80 psi, 34.4 gpm), pneumatic_cylinder (5.0 in_bore, 2.0 in_rod, 40.2 ft/min, 16 ft_stroke))] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

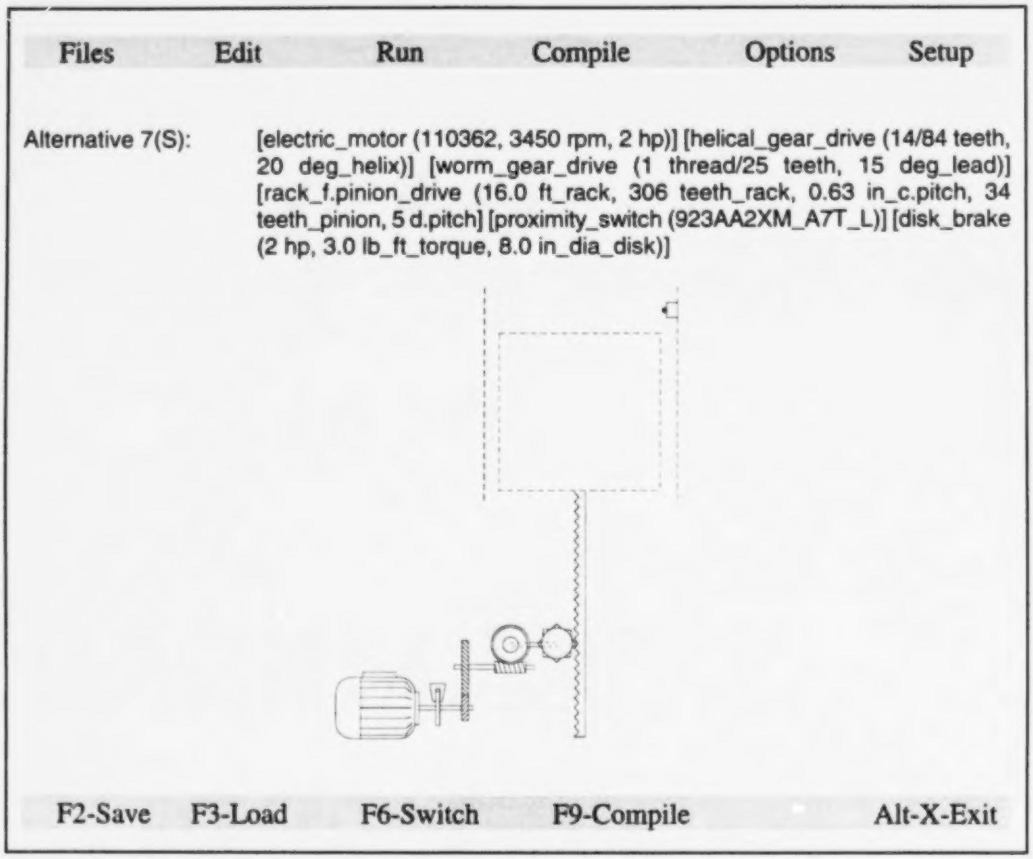
Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 6(S): [electric_motor (110362, 3450 rpm, 2 hp)] [pneumatic_drive (vane_pump (3450 rpm, 100 psi, 26.2 gpm), pneumatic_cylinder (6.0 in_bore, 2.0 in_rod, 20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8x19-0.5 in_rope, 22 in_dia_pulley, 0.27 in_groove-radius)] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

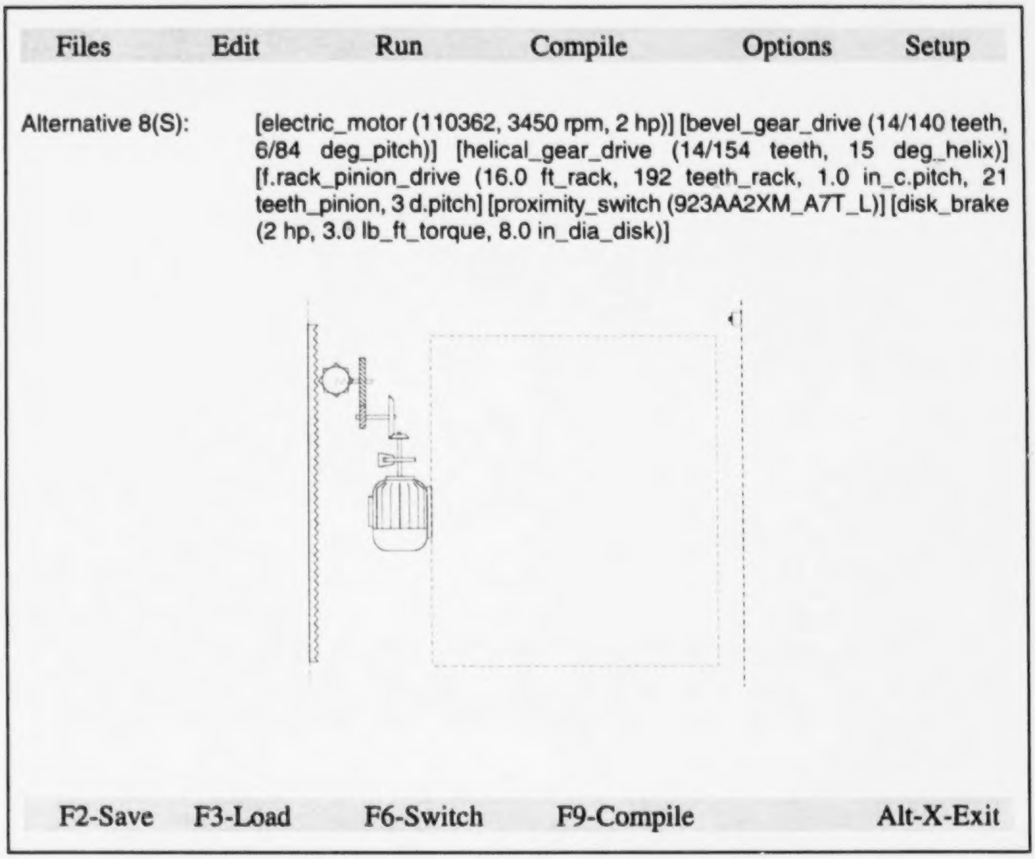
Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 7(S): [electric_motor (110362, 3450 rpm, 2 hp)] [helical_gear_drive (14/84 teeth, 20 deg_helix)] [worm_gear_drive (1 thread/25 teeth, 15 deg_lead)] [rack_f.pinion_drive (16.0 ft_rack, 306 teeth_rack, 0.63 in_c.pitch, 34 teeth_pinion, 5 d.pitch] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

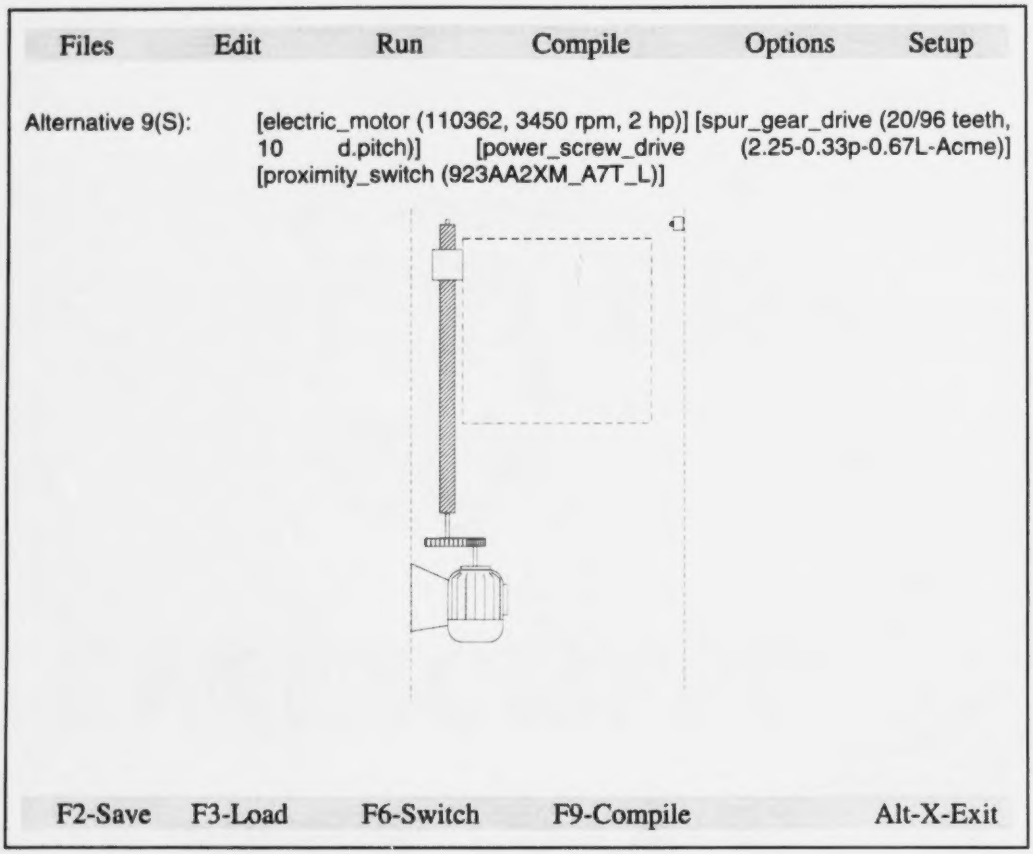
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Alternative 8(S): [electric_motor (110362, 3450 rpm, 2 hp)] [bevel_gear_drive (14/140 teeth, 6/84 deg_pitch)] [helical_gear_drive (14/154 teeth, 15 deg_helix)] [f.rack_pinion_drive (16.0 ft_rack, 192 teeth_rack, 1.0 in_c.pitch, 21 teeth_pinion, 3 d.pitch] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

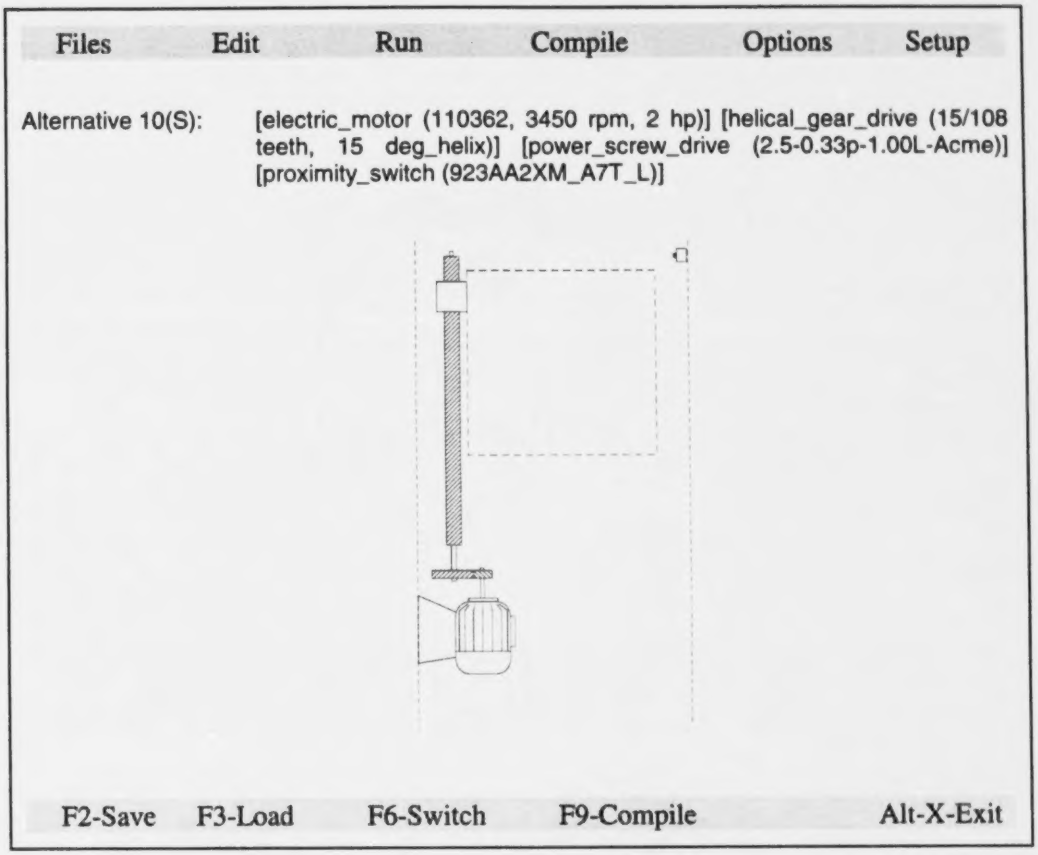
Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 9(S): [electric_motor (110362, 3450 rpm, 2 hp)] [spur_gear_drive (20/96 teeth, 10 d.pitch)] [power_screw_drive (2.25-0.33p-0.67L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

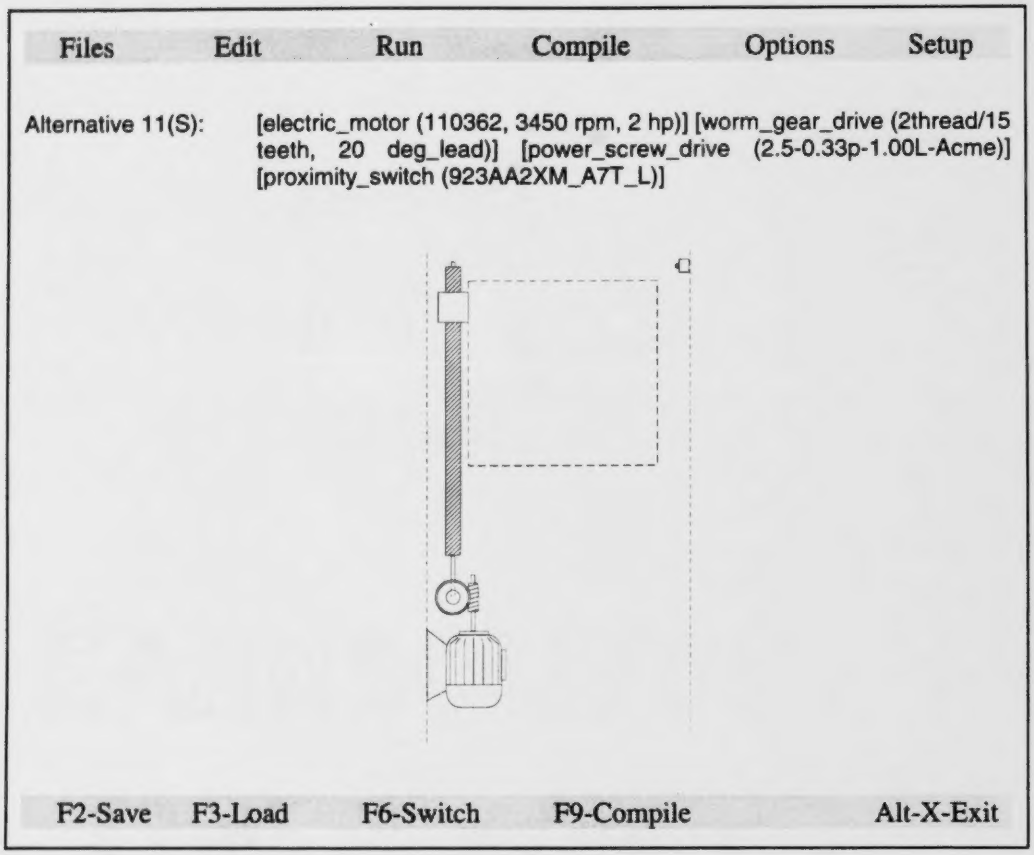
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

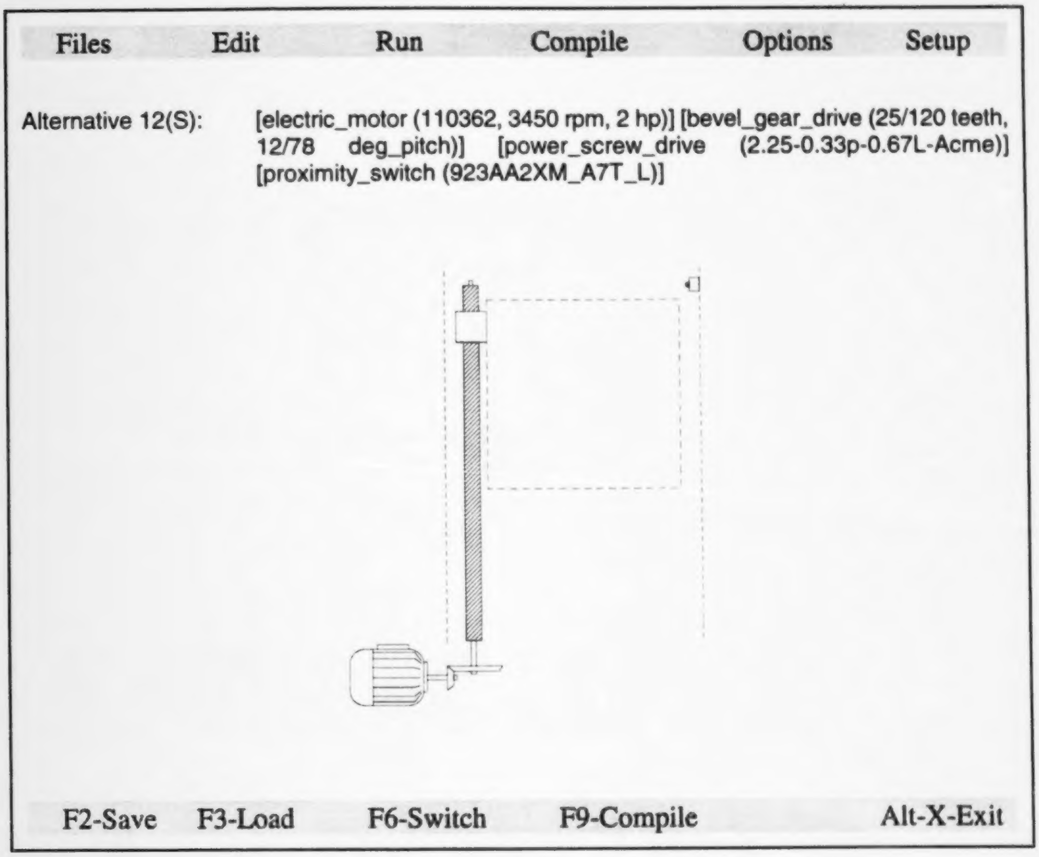
Alternative 10(S): [electric_motor (110362, 3450 rpm, 2 hp)] [helical_gear_drive (15/108 teeth, 15 deg_helix)] [power_screw_drive (2.5-0.33p-1.00L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

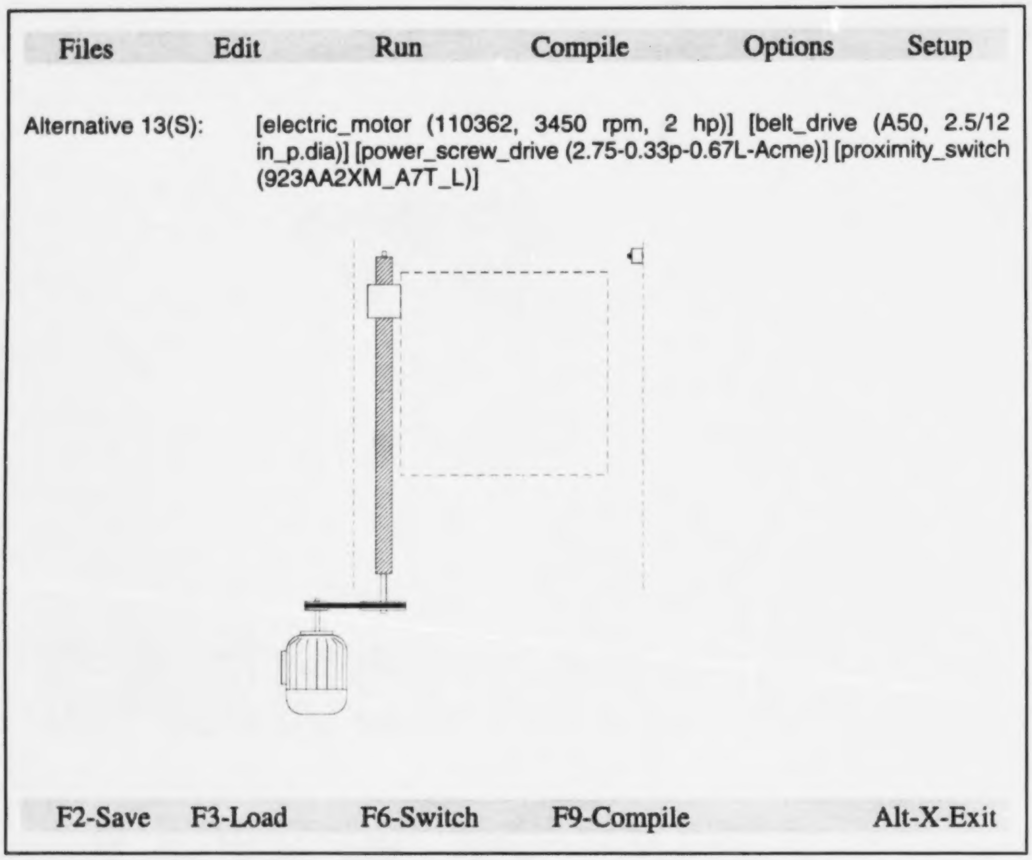
Alternative 11(S): [electric_motor (110362, 3450 rpm, 2 hp)] [worm_gear_drive (2thread/15 teeth, 20 deg_lead)] [power_screw_drive (2.5-0.33p-1.00L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

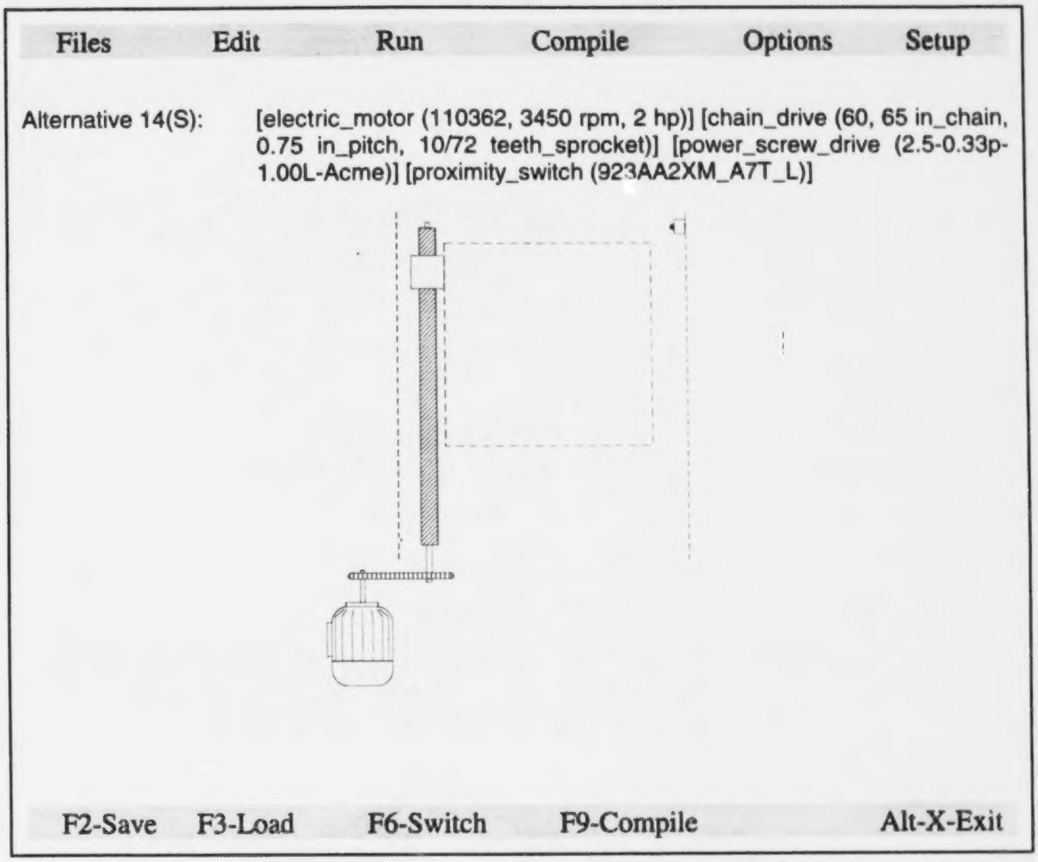
Alternative 12(S): [electric_motor (110362, 3450 rpm, 2 hp)] [bevel_gear_drive (25/120 teeth, 12/78 deg_pitch)] [power_screw_drive (2.25-0.33p-0.67L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

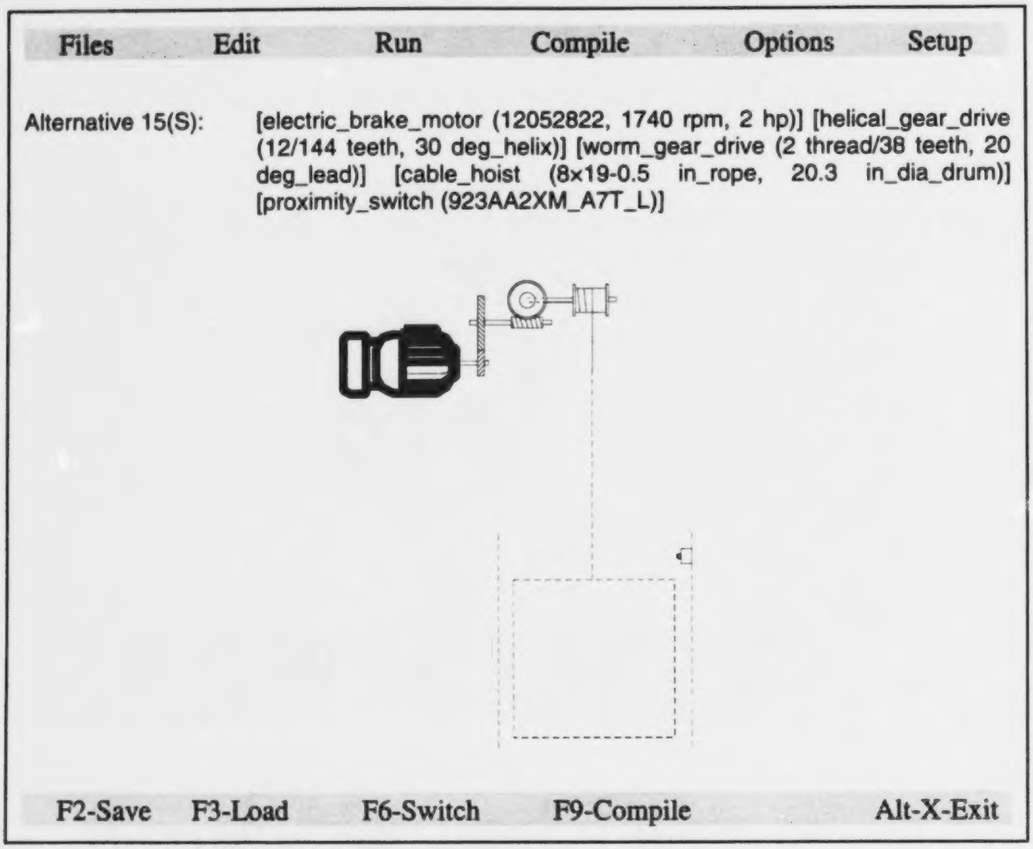
Alternative 13(S): [electric_motor (110362, 3450 rpm, 2 hp)] [belt_drive (A50, 2.5/12 in_p.dia)] [power_screw_drive (2.75-0.33p-0.67L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

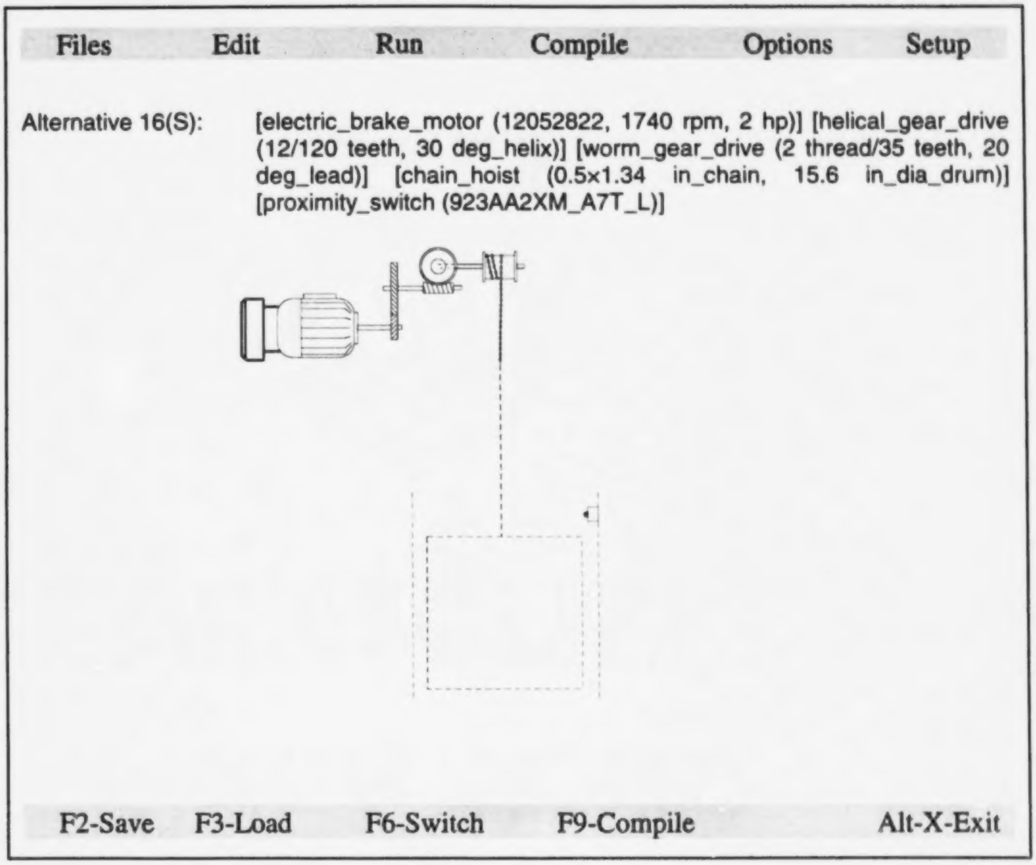
Alternative 14(S): [electric_motor (110362, 3450 rpm, 2 hp)] [chain_drive (60, 65 in_chain, 0.75 in_pitch, 10/72 teeth_sprocket)] [power_screw_drive (2.5-0.33p-1.00L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Alternative 15(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [helical_gear_drive (12/144 teeth, 30 deg_helix)] [worm_gear_drive (2 thread/38 teeth, 20 deg_lead)] [cable_hoist (8x19-0.5 in_rope, 20.3 in_dia_drum)] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)

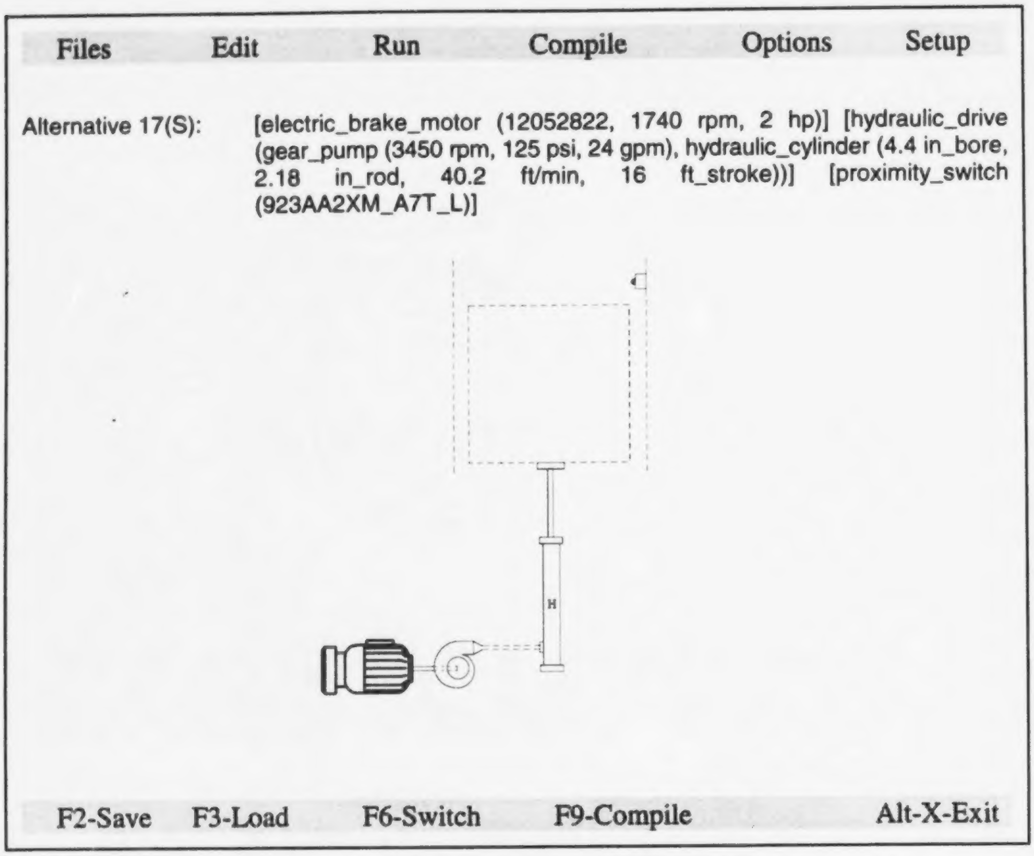


Files
Edit
Run
Compile
Options
Setup

Alternative 16(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [helical_gear_drive (12/120 teeth, 30 deg_helix)] [worm_gear_drive (2 thread/35 teeth, 20 deg_lead)] [chain_hoist (0.5×1.34 in_chain, 15.6 in_dia_drum)] [proximity_switch (923AA2XM_A7T_L)]

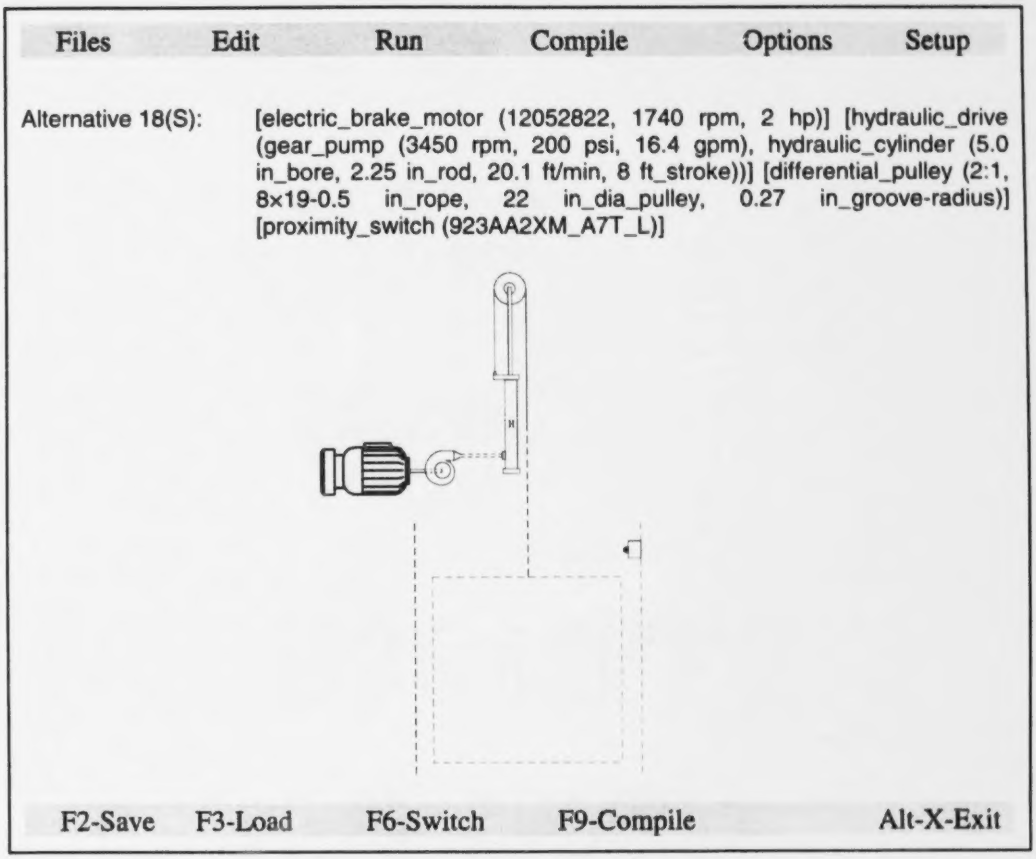
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
H
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

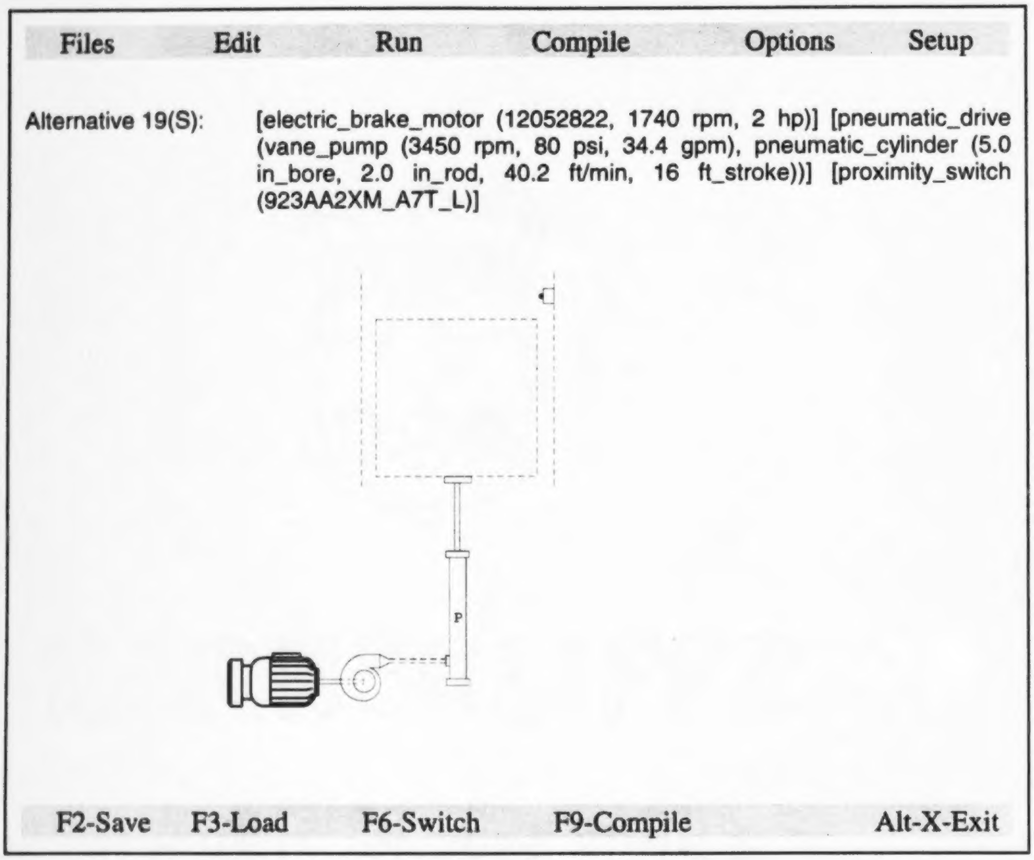
Alternative 17(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [hydraulic_drive (gear_pump (3450 rpm, 125 psi, 24 gpm), hydraulic_cylinder (4.4 in_bore, 2.18 in_rod, 40.2 ft/min, 16 ft_stroke))] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

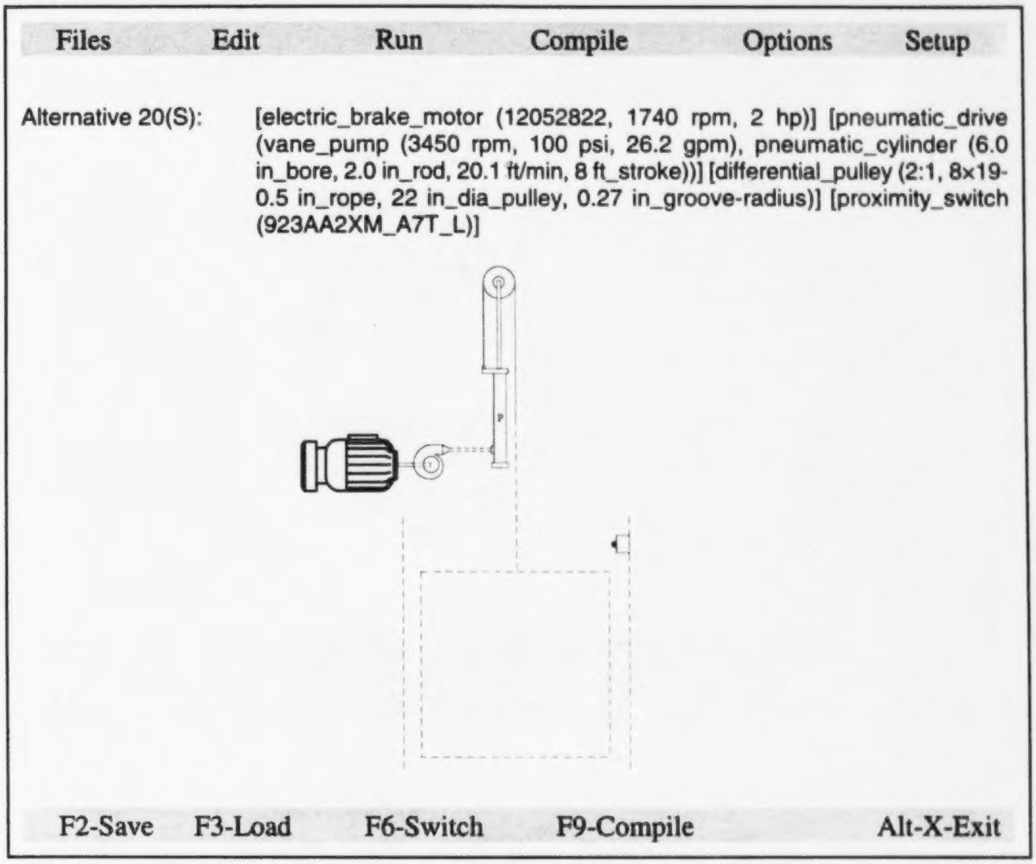
Alternative 18(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [hydraulic_drive (gear_pump (3450 rpm, 200 psi, 16.4 gpm), hydraulic_cylinder (5.0 in_bore, 2.25 in_rod, 20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8×19-0.5 in_rope, 22 in_dia_pulley, 0.27 in_groove-radius)] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
P
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

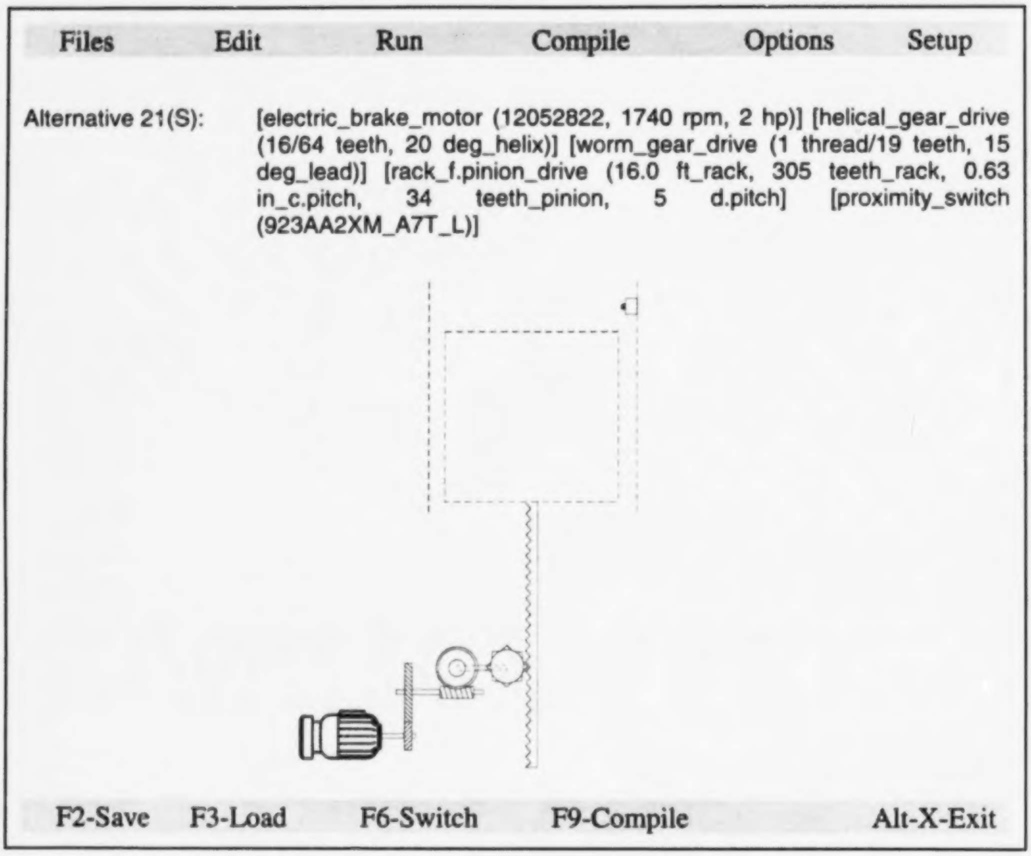
Alternative 19(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [pneumatic_drive (vane_pump (3450 rpm, 80 psi, 34.4 gpm), pneumatic_cylinder (5.0 in_bore, 2.0 in_rod, 40.2 ft/min, 16 ft_stroke))] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

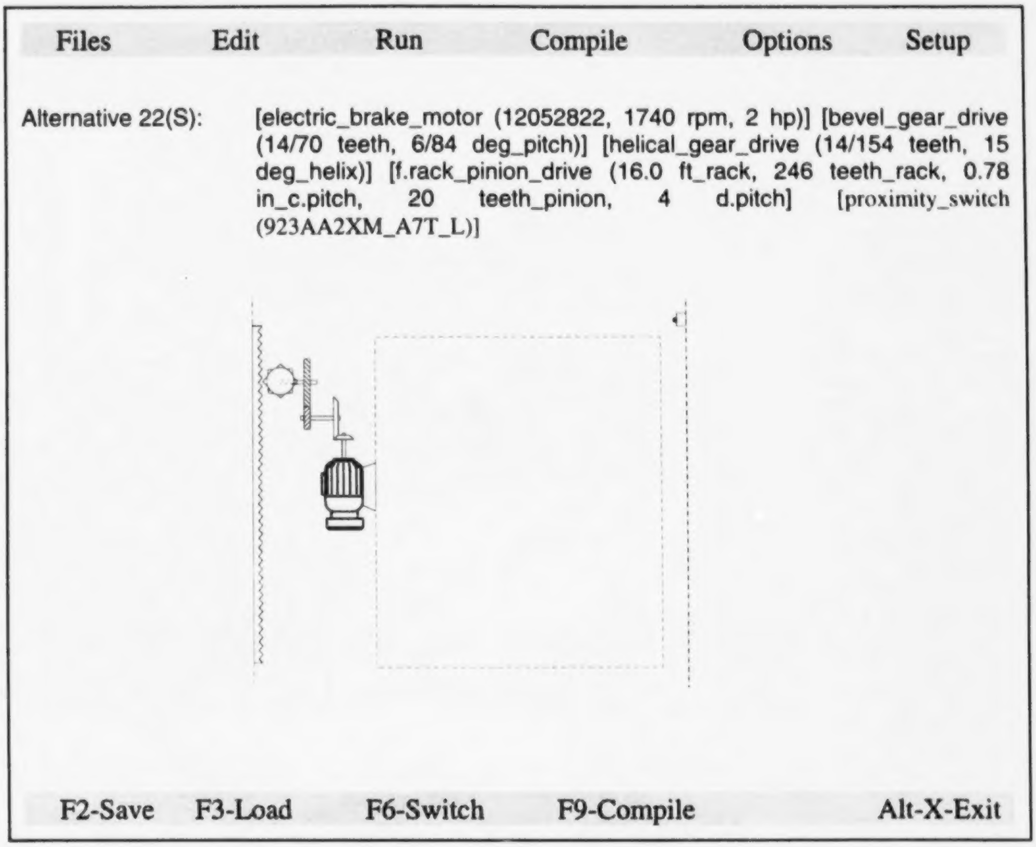
Alternative 20(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [pneumatic_drive (vane_pump (3450 rpm, 100 psi, 26.2 gpm), pneumatic_cylinder (6.0 in_bore, 2.0 in_rod, 20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8x19-0.5 in_rope, 22 in_dia_pulley, 0.27 in_groove-radius)] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

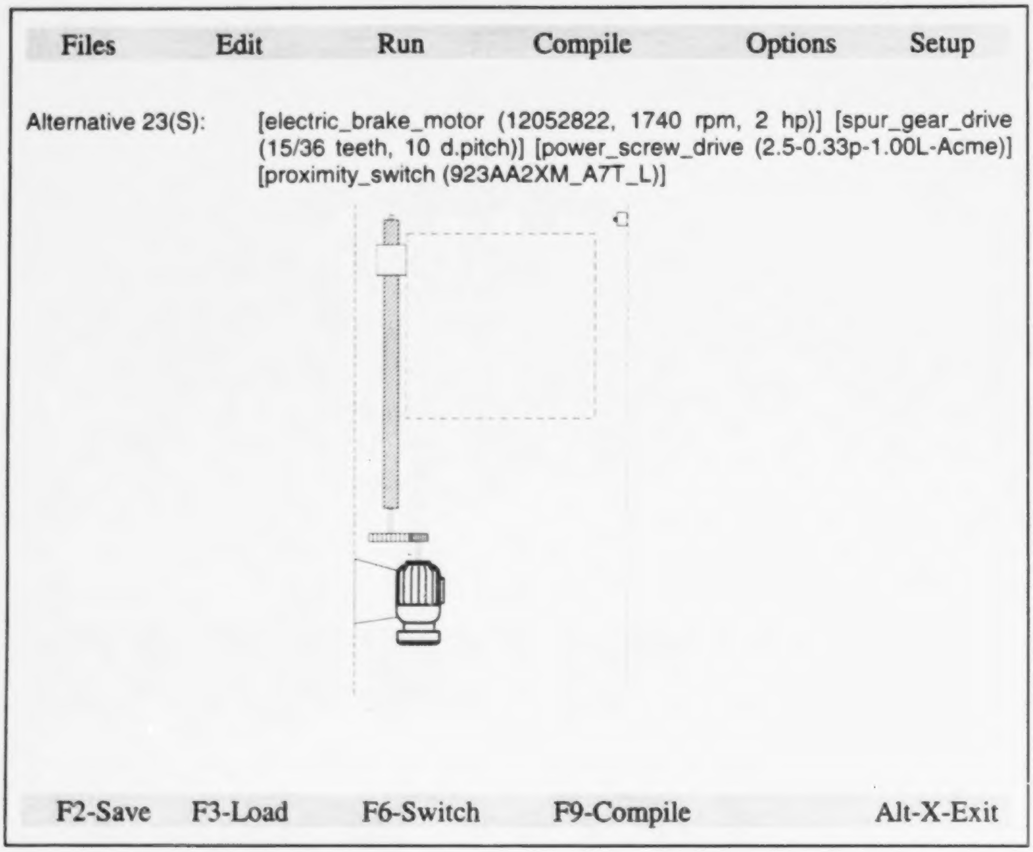
Alternative 21(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [helical_gear_drive (16/64 teeth, 20 deg_helix)] [worm_gear_drive (1 thread/19 teeth, 15 deg_lead)] [rack_f.pinion_drive (16.0 ft_rack, 305 teeth_rack, 0.63 in_c.pitch, 34 teeth_pinion, 5 d.pitch] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

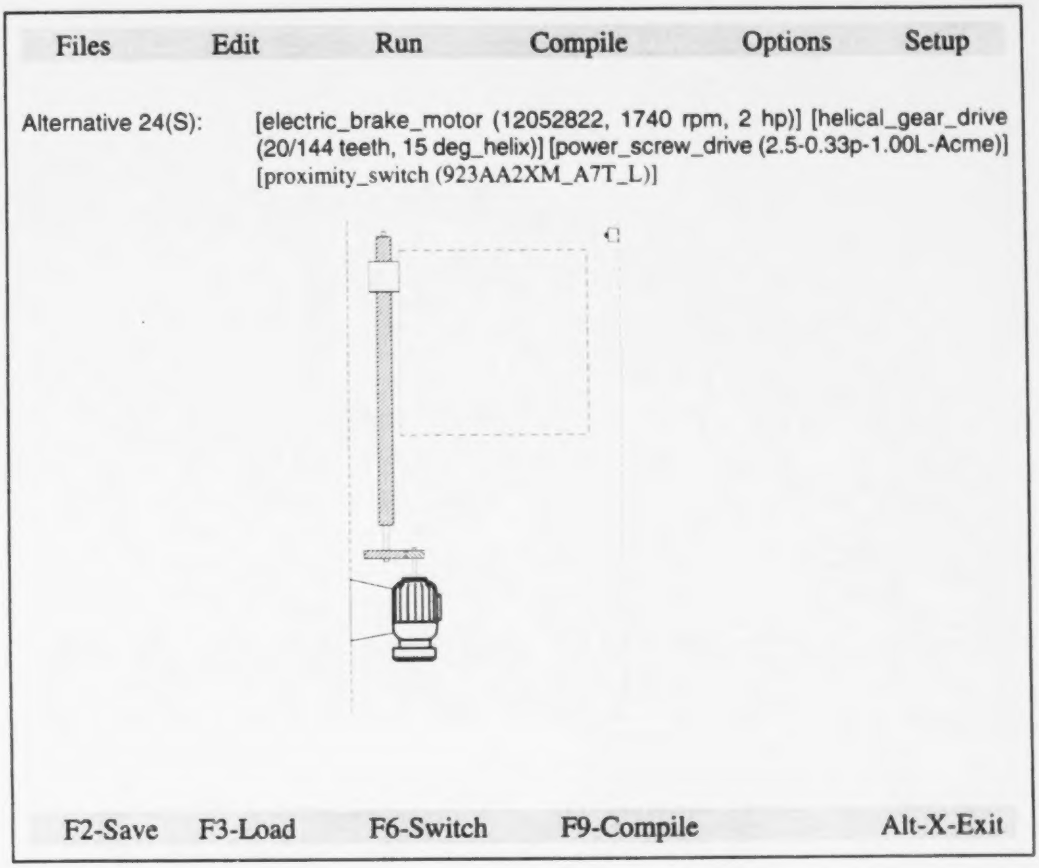
Alternative 22(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [bevel_gear_drive (14/70 teeth, 6/84 deg_pitch)] [helical_gear_drive (14/154 teeth, 15 deg_helix)] [f.rack_pinion_drive (16.0 ft_rack, 246 teeth_rack, 0.78 in_c.pitch, 20 teeth_pinion, 4 d.pitch] [proximity_switch (923AA2XM_A7T_L)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Alternative 23(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [spur_gear_drive (15/36 teeth, 10 d.pitch)] [power_screw_drive (2.5-0.33p-1.00L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

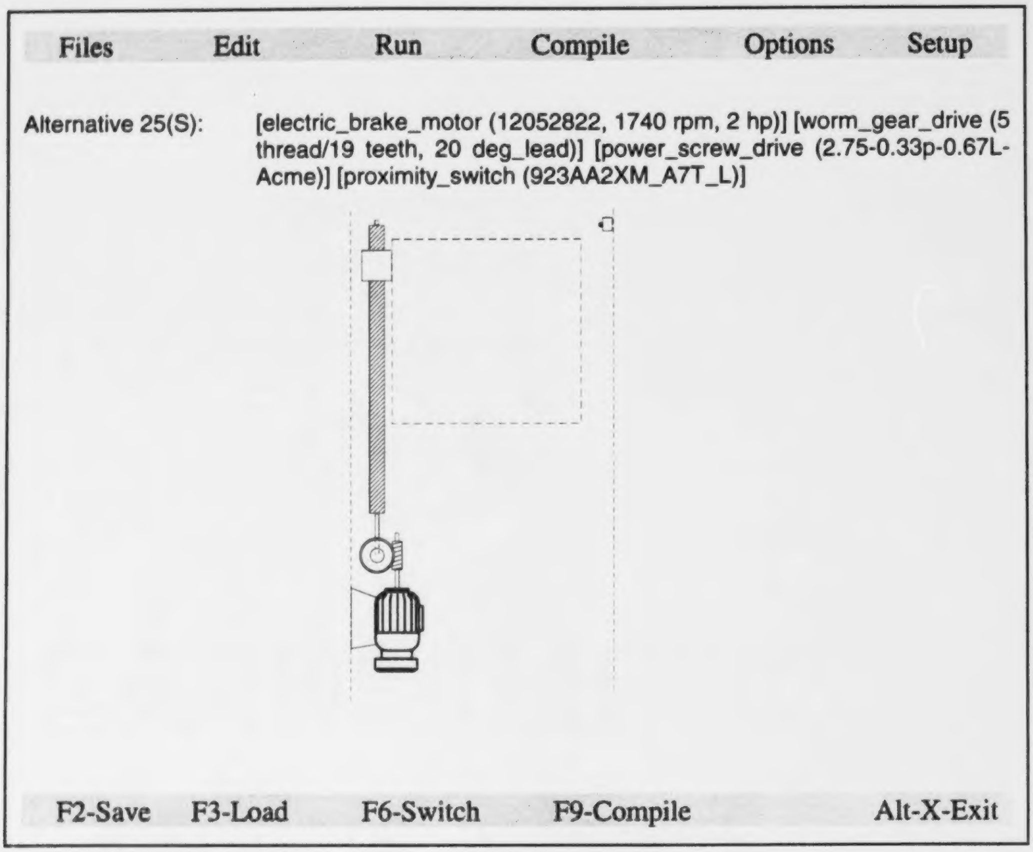
Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 24(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [helical_gear_drive (20/144 teeth, 15 deg_helix)] [power_screw_drive (2.5-0.33p-1.00L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

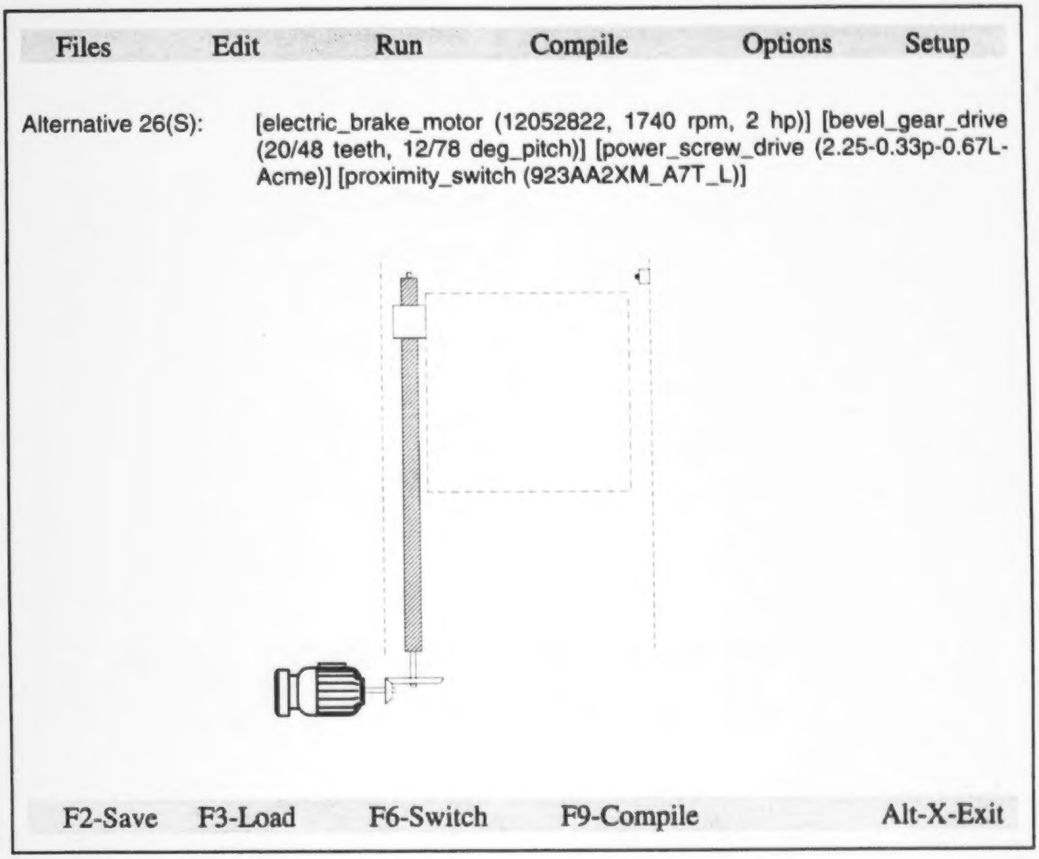
Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 25(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [worm_gear_drive (5 thread/19 teeth, 20 deg_lead)] [power_screw_drive (2.75-0.33p-0.67L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

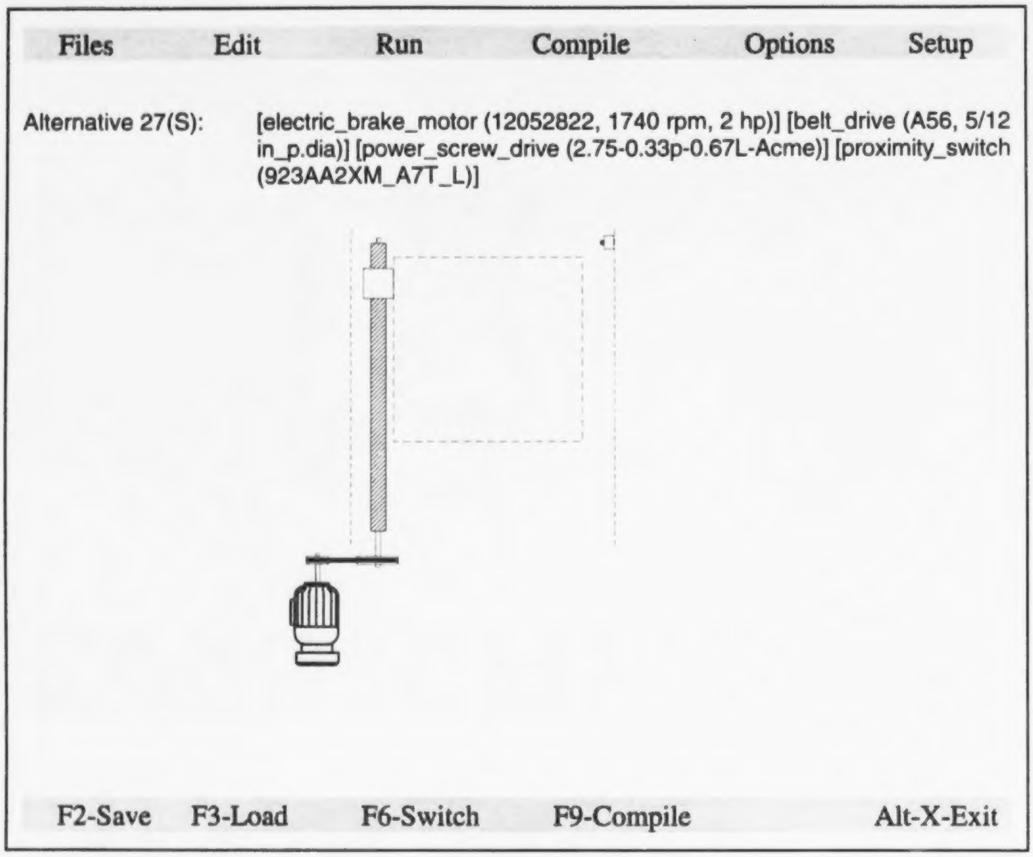
Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 26(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [bevel_gear_drive (20/48 teeth, 12/78 deg_pitch)] [power_screw_drive (2.25-0.33p-0.67L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

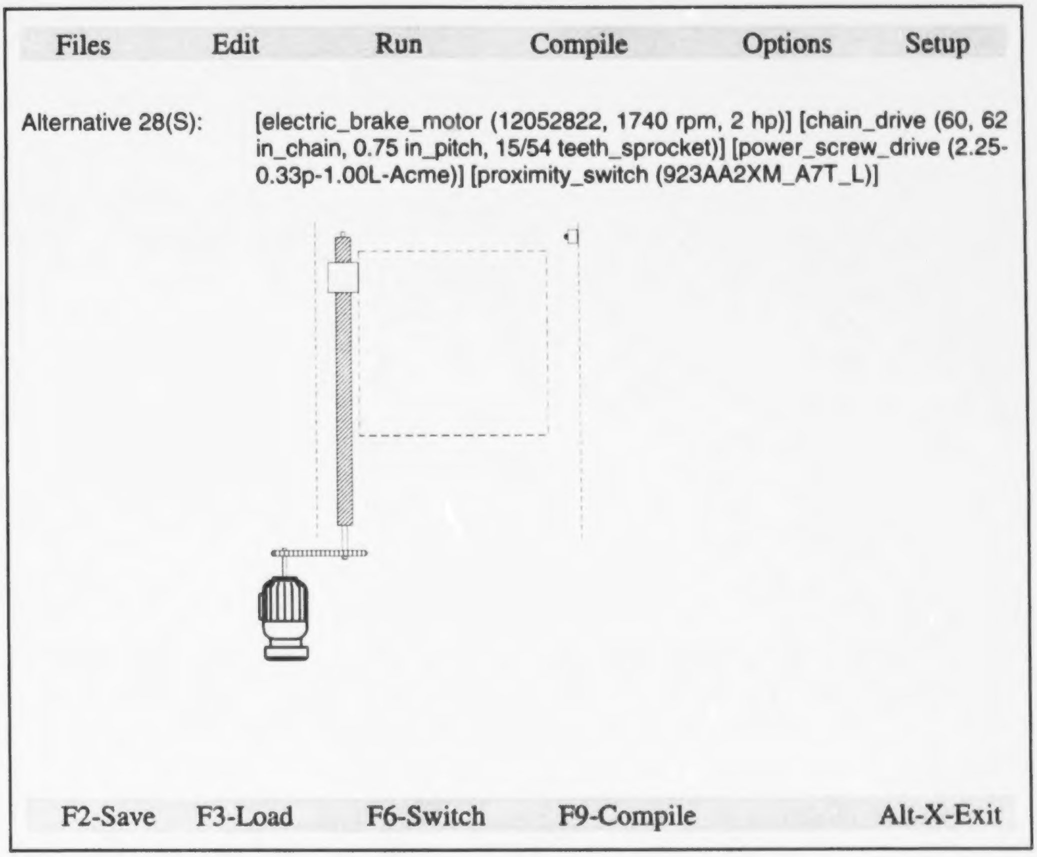
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Alternative 27(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [belt_drive (A56, 5/12 in_p.dia)] [power_screw_drive (2.75-0.33p-0.67L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

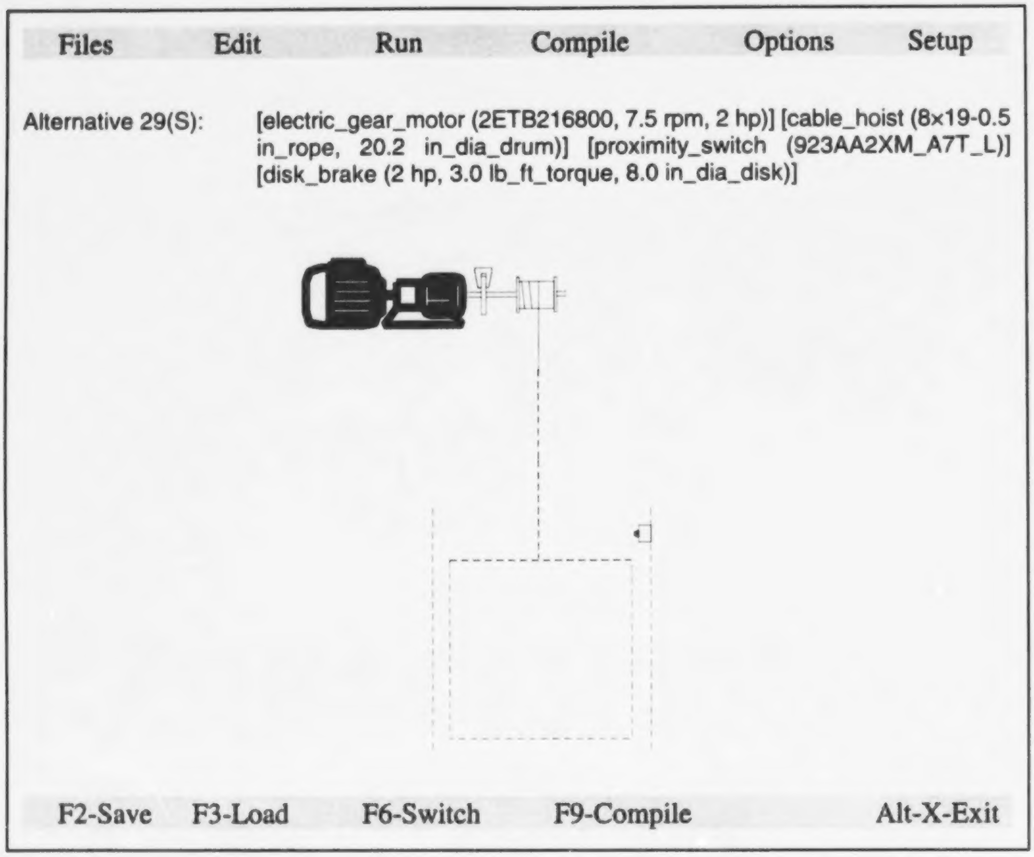
Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 28(S): [electric_brake_motor (12052822, 1740 rpm, 2 hp)] [chain_drive (60, 62 in_chain, 0.75 in_pitch, 15/54 teeth_sprocket)] [power_screw_drive (2.25-0.33p-1.00L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

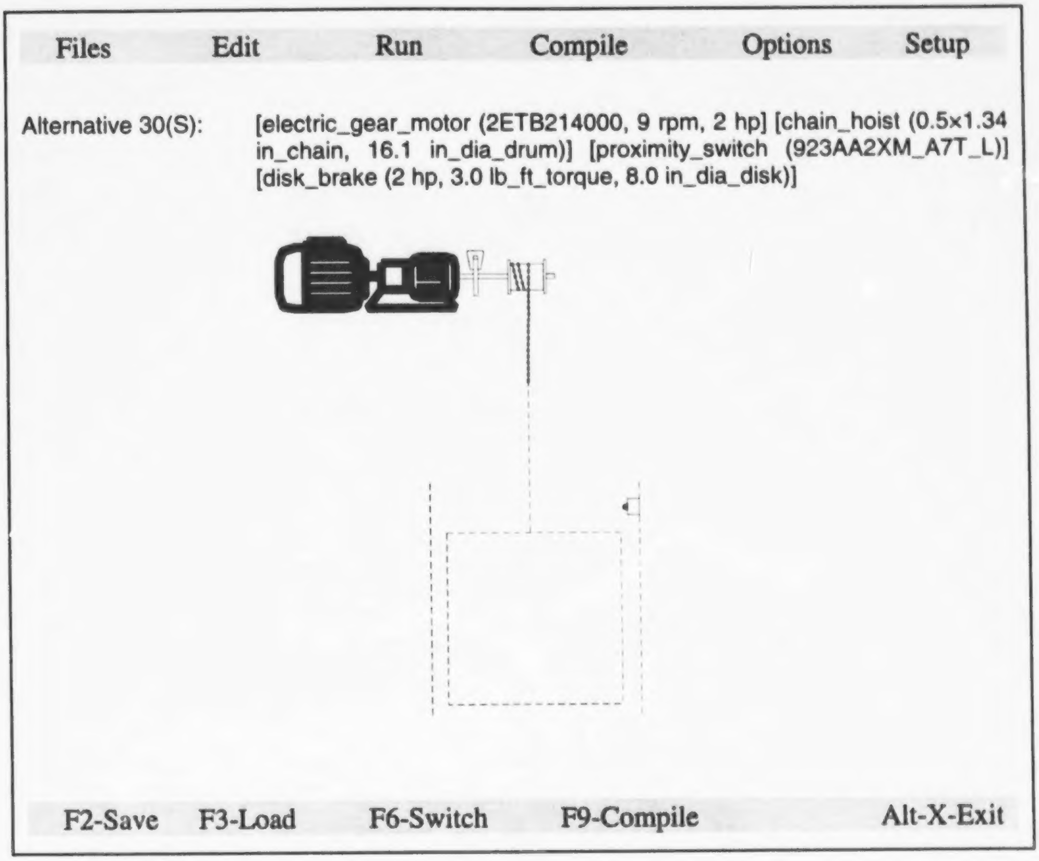
Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 29(S): [electric_gear_motor (2ETB216800, 7.5 rpm, 2 hp)] [cable_hoist (8x19-0.5 in_rope, 20.2 in_dia_drum)] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

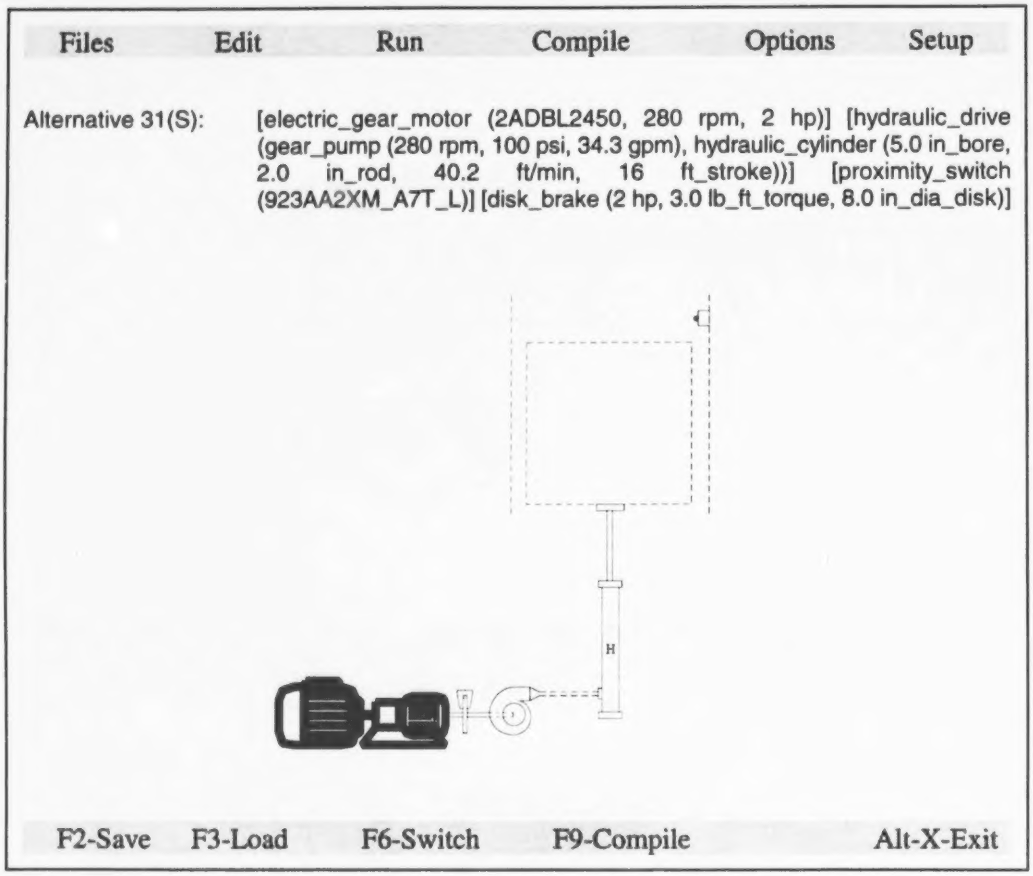
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

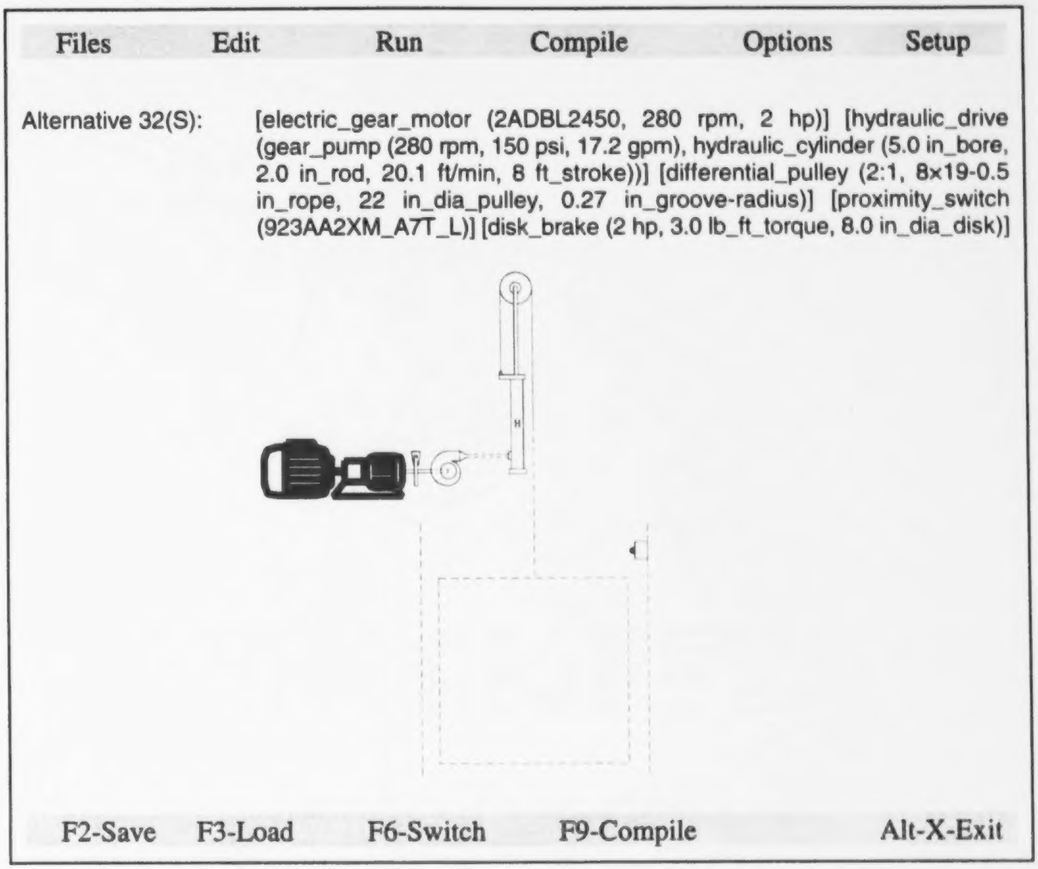
Alternative 30(S): [electric_gear_motor (2ETB214000, 9 rpm, 2 hp] [chain_hoist (0.5×1.34 in_chain, 16.1 in_dia_drum)] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
H
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

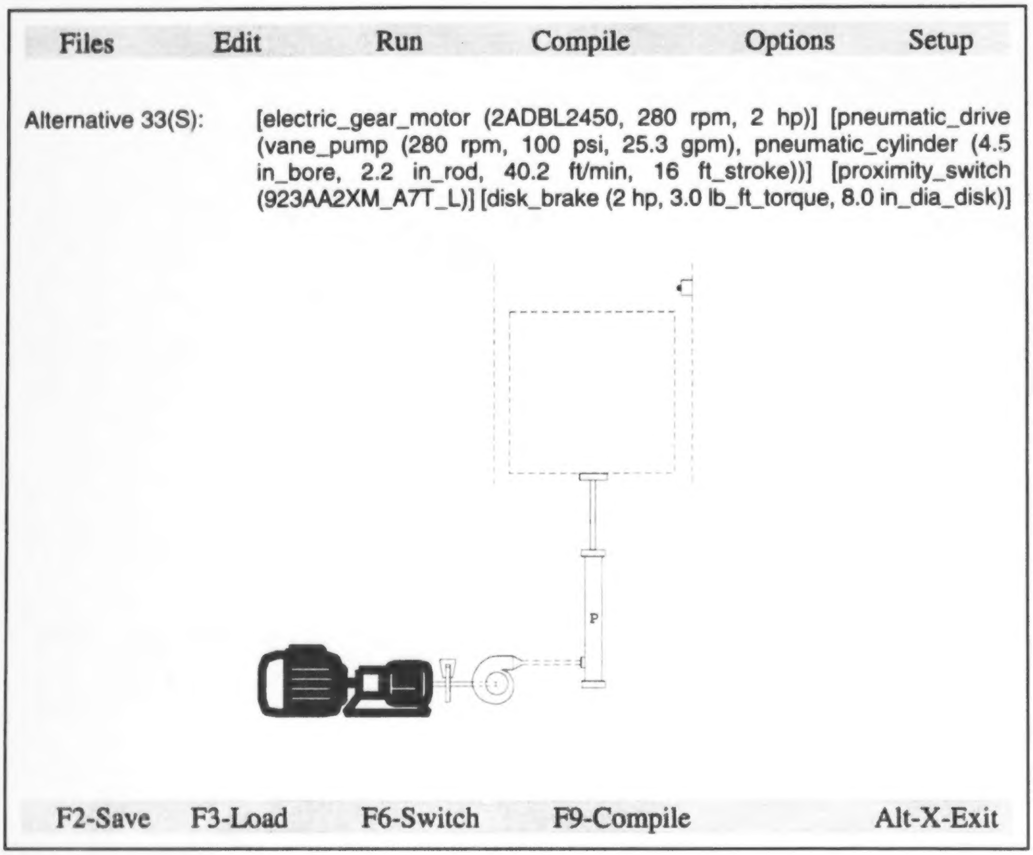
Alternative 31(S): [electric_gear_motor (2ADBL2450, 280 rpm, 2 hp)] [hydraulic_drive (gear_pump (280 rpm, 100 psi, 34.3 gpm), hydraulic_cylinder (5.0 in_bore, 2.0 in_rod, 40.2 ft/min, 16 ft_stroke))] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

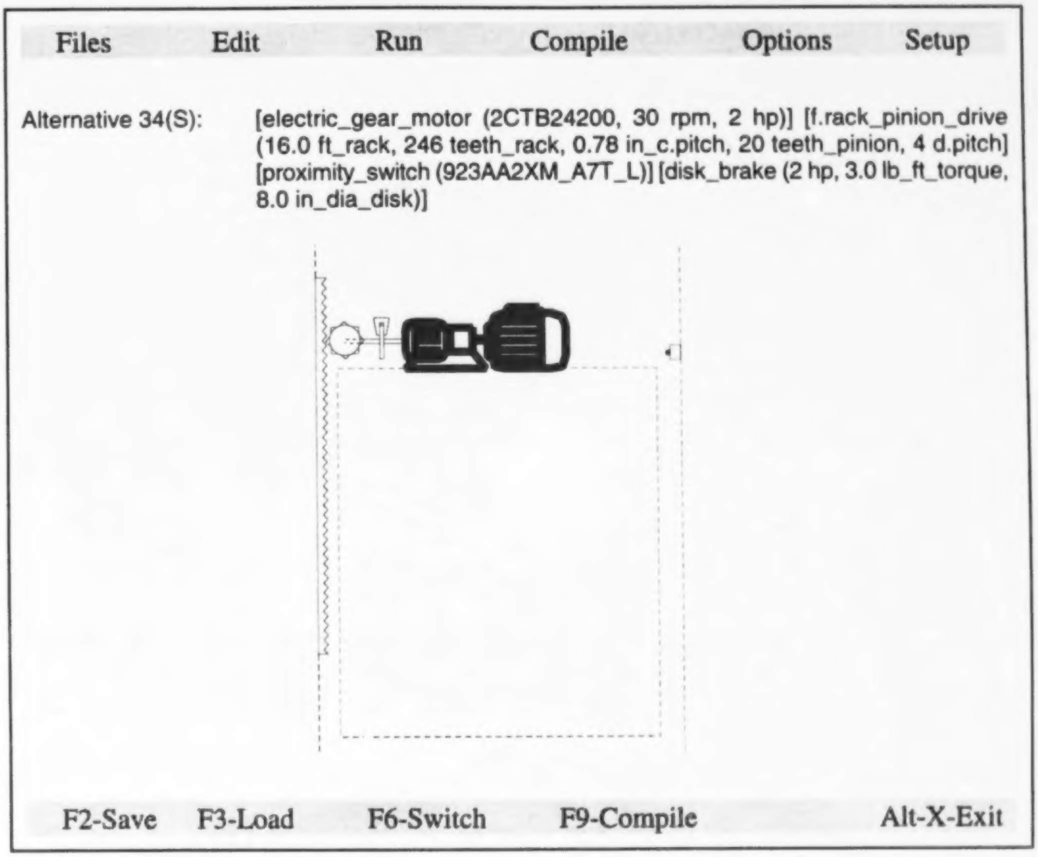
Alternative 32(S): [electric_gear_motor (2ADBL2450, 280 rpm, 2 hp)] [hydraulic_drive (gear_pump (280 rpm, 150 psi, 17.2 gpm), hydraulic_cylinder (5.0 in_bore, 2.0 in_rod, 20.1 ft/min, 8 ft_stroke))] [differential_pulley (2:1, 8×19-0.5 in_rope, 22 in_dia_pulley, 0.27 in_groove-radius)] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup
P
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Alternative 33(S): [electric_gear_motor (2ADBL2450, 280 rpm, 2 hp)] [pneumatic_drive (vane_pump (280 rpm, 100 psi, 25.3 gpm), pneumatic_cylinder (4.5 in_bore, 2.2 in_rod, 40.2 ft/min, 16 ft_stroke))] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

Figure 6-23: Final design alternatives (Continued)

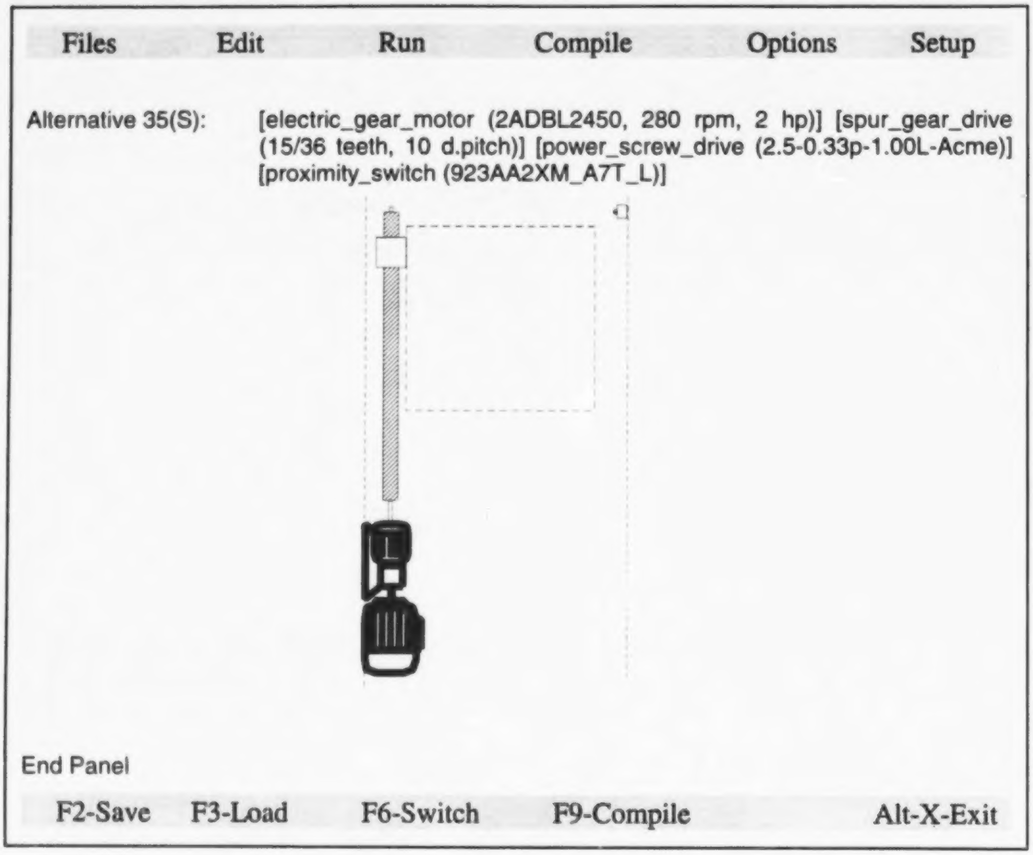


Files
Edit
Run
Compile
Options
Setup

Alternative 34(S): [electric_gear_motor (2CTB24200, 30 rpm, 2 hp)] [f.rack_pinion_drive (16.0 ft_rack, 246 teeth_rack, 0.78 in_c.pitch, 20 teeth_pinion, 4 d.pitch] [proximity_switch (923AA2XM_A7T_L)] [disk_brake (2 hp, 3.0 lb_ft_torque, 8.0 in_dia_disk)]

F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Figure 6-23: Final design alternatives (Continued)


Files
Edit
Run
Compile
Options
Setup

Alternative 35(S): [electric_gear_motor (2ADBL2450, 280 rpm, 2 hp)] [spur_gear_drive (15/36 teeth, 10 d.pitch)] [power_screw_drive (2.5-0.33p-1.00L-Acme)] [proximity_switch (923AA2XM_A7T_L)]

End Panel

F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

Figure 6-23: Final design alternatives (Concluded)

CHAPTER 7

SUMMARY AND CONCLUSIONS

In this chapter we present the essence of the research reported here and the foundations on which it has been built. We shall not, however, go into details of the points outlined here, as they have been elaborated in the body of the work.

We started off by distinguishing between *design* and *design methodology*, and described the latter as the answer to the question "how to design" (that is, how the design process does, or should, take place) rather than "what to design". We then discussed the major paradigms intended to set forth the basic steps of the design process in general.

Having recognized the main stages of the mechanical design process as *functional-*, *conceptual-* and *parametric-design*, we then explained how well each stage has so far been explored and discussed some of the complications that have impeded a thorough understanding and computer-implementation (automation) of the entire process.

We asserted that the process of mechanical design in its entirety may not be automated until, for each and every stage of the process, a formal model is developed that would suggest a clear, logical and realistic scheme for implementing that stage, and that could be implemented in computer code.

This assertion plus the observation that by far only the third stage of the mechanical design process (i.e. parametric design) has been fairly well modelled and implemented made us realize the importance and urgency of developing computer-based models for conceptual- and functional-design stages.

Of these two stages mentioned above, conceptual design was the logical choice at this time because:

- The stage was bounded from below by the well-established parametric design, meaning that we knew the expectations and implications of the following stage on the current one (e.g. the need for parametric consistency), and we would be able to exploit this knowledge to avoid post-development inconsistencies between the two stages. Furthermore, there was an overlap between the two stages, as they were both concerned with the examination of the components' constraint sets. And
- Conceptual design seemed implementable with the existing computer technology, whereas functional design, because of its highly abstract nature, was, and still is, beyond the reach of existing AI technology.

A survey of the literature on computer-based models of mechanical conceptual design showed that the very few models proposed in recent years either are highly case-dependent or are still at a theoretical level and lack a practical implementation scheme. This motivated us to focus on the development and computer implementation of a model that can be applied to a relatively large class of mechanical systems.

In this work, we have presented a computer-based model for mechanical conceptual design called Design by Exploration. The model builds on three main assertions, viz.

1. Mechanical systems *can* be conceptually designed through a stepwise transformation of their functional description to structural description, provided that the former is expressed in terms of a set of standard functions, and that those functions can be mapped, individually or collectively, onto existing physical components.
2. A blind, one-to-one mapping of functions to physical components will result in poor or even infeasible designs. Each mapping should be supported by an exploration of the component's functional behavior. The constraint set of the component must be solved before its contribution to the overall functioning of the system can be fully determined.
3. A best-first search of the solution space is likely to result in the loss of the optimum design and/or some of the feasible designs. Wherever possible, all design alternatives should be considered so that the system could generate more design ideas and preserve them for a final selection stage.

Design by Exploration has been implemented in the context of *Conceptual Designer*, a design environment which would allow the user to introduce a design problem in terms of its desired functions plus any applicable qualitative/quantitative constraints, and obtain a number (≥ 1) of feasible solutions to the problem simultaneously.

We acknowledge the fact that the function of a mechanical system is affected, one way or another, by the environment that surrounds it, and that in the assessment of the "feasibility" and "goodness" of a system, one should consider the entire lifecycle of the system as well as its environment. This acknowledgement obliges us to take into account not only the specified functional requirements but also the implications of other lifecycle objectives (such as manufacturability, cost, etc.) as well as the environment.

To accommodate this, Conceptual Designer has been designed based on a blackboard paradigm of problem solving. The paradigm allows multiple expert modules, representing various lifecycle objectives and the environment, to contribute to the design process. This enables the exploitation of the expert knowledge of experienced designers in an automated design system; a feature that most such design systems lack.

Earlier we mentioned "universality" or "domain-independence" as an important characteristic of a good design system. To implement this in Conceptual Designer, we have employed such universal (domain-independent) computational techniques as the Genetic Algorithms to make it possible for the system to handle problems of virtually any initial specifications with minimum or no need for user intervention.

Nevertheless, one should bear in mind that neither Design by Exploration nor its implementation, the Conceptual Designer, are meant as stand-alone entities. They form a middle link in the three-link chain of mechanical design which only collectively can perform the design process in its entirety. DbE relies heavily on the information generated at the Functional Design stage, and in turn produces the essential information for the next stage, i.e. Parametric Design.

The *instantiation* scheme developed as part of the DbE's exploration stage can also be regarded as a universal parametric-design scheme and can be implemented into a separate package. Whenever required, we have articulated our reasons for choosing the computational and search techniques that we have employed in this scheme. The reader should note that the implementation scheme presented in this work is only one, possibly out of several, ways to implement the proposed model. Whereas we believe that it is a robust scheme, we do not rule out the possibility of other schemes being developed now or with the emergence of new computational techniques.

The DbE model has been successfully applied to the design of several systems/sub-systems, including the one presented in Chapter 6. However, there are, of necessity, limitations to its capabilities and the type of problems it can solve. These limitations have been highlighted in previous Chapters, especially in Chapters 5 and 6. It is our future goal to try to overcome these limitations and to develop a greater computer system for carrying out the entire process of mechanical design.

To summarize, the main contributions of this thesis are as follows:

1. A computer-based model of mechanical conceptual design has been proposed which, unlike some of the existing models, is not inspired by the models of electrical- or software engineering, and hence can be applied to mechanical design problems more accurately and efficiently. The model is superior to the existing models in three major ways:
    a. It prescribes the "instantiation" of the selected components *during* the function-to-embodiment transformation rather than after it's completion. This will allow the specific characteristics of mechanical elements (such as multifunctionality and form-function relations) to be taken into consideration in the design process and will therefore result in more feasible and more realistic solutions.
    b. It prescribes the preservation of virtually all feasible solutions generated at each

stage of the process (as opposed to a best-first strategy) and thus greatly decreases the chance of the optimal solution being lost or overlooked.

   c. It allows for the consideration of *constraints* (both "initially specified" and "system generated") in addition to the functional requirements, and therefore can be applied to real-world problems.

2. A comprehensive implementation scheme for the proposed model has been developed which is virtually user-independent and insensitive to the case (the product to be designed) and the initial specifications of the problem. It also accommodates the notion of concurrent design, that is, the implications of the product's various lifecycle objectives and the environment are taken into account in the design process. The scheme has been, for the most part, implemented in code and successfully tested on a variety of problems.
3. A new approach to parametric design has been presented based on the use of Genetic Algorithms. The use of GAs enables the system to process numerical and certain non-numerical constraints simultaneously. The presented approach is superior to the calculus-based "constraint management" approaches in that, among other things, it does not require knowledge of mathematical nature of the constraints (e.g. continuity and differentiability).

The development of so-called intelligent systems that can possibly replace human designers or even accurately mimic them in their design activities is still far from reality. We do not claim, nor have we ever claimed, that we have achieved this goal with the research presented here. However, we believe that with this work we have achieved a better understanding of the nature of this highly intelligent activity of human mind and have contributed to the development of systems that will reduce the burden of routine design so that designers can devote their creativity to more challenging tasks. This by itself is an ambitious goal one cannot leap to, and has to be approached incrementally. The current research is one step towards this goal.

BIBLIOGRAPHY

Adelson, B. 1989 (a). Modelling Software Design Within a Problem Solving Architecture. *Design Theory 88: Proceedings of the 1988 NSF Workshop on Design Theory and Methodology*, Springer-Verlag, New York.

Adelson, B. 1989 (b). Cognitive Research: Uncovering How Designers Design; Cognitive Modeling: Explaining and Predicting How Designers Design, Research in Engineering Design, Vol. 1, No. 1.

Adelson, B. and Soloway, E. 1984. A Cognitive Model of Software Design. *Report 342*, Department of Computing Sciences, Yale.

Agogino, A.M. and Almgren, A.S. 1987a. Symbolic Computation in Computer-Aided Optimal Design. *Expert Systems in Computer-Aided Design*. Gero, J.S. (Ed.), North-Holland, Amsterdam, pp 267-284.

Agogino, A.M. and Almgren, A.S. 1987b. Techniques for Integrating Qualitative Reasoning and Symbolic Computation in Engineering Optimization. *Engineering Optimization*. Vol.12, No.2, pp 117-135.

Agrawal, R., Kinzel, G.L., Srinivasan, R.V. and Ishii, K. 1993. Engineering Constraint Management Based on an Occurrence Matrix Approach. *Journal of Mechanical Design*. ASME, Vol. 115, No. 1, pp 103-109.

Arora, J.S. 1989. *Introduction to Optimum Design*. McGraw-Hill.

Arora, J.S. and Baenziger, G. 1986. Use of Artificial Intelligence in Design Optimization.

Computer Methods in Applied Mechanics and Engineering. Vol.54, pp 303-323.

Asimow, M. 1962. *Introduction to Design*. Prentice-Hall, New York.

Aylmer, L. and Johnson, M.A. 1987. Function Modelling: A New Development in Computer Aided Design. *Intelligent CAD II*. Yoshikawa, H. and Holton, T. (Ed.), North-Holland.

Azarm, S. and Papalambros, P. 1984. An Automated Procedure for Local Monotonicity Analysis. *Journal of Mechanisms, Transmissions and Automation in Design*. Vol.106, No.1, pp 82-89.

Balachandran, M. and Gero, J.S. 1987. A Knowledge-Based Approach to Mathematical Design Modelling and Optimization. *Engineering Optimization*. Vol.12, pp 91-115.

Baumeister, T. 1978. Standard Handbook for Mechanical Engineers., 8th Ed. McGraw-Hill.

Beasley, D., Bull, D.R. and Martin, R.R. 1993. An Overview of Genetic Algorithms: Part I, Fundamentals. *University computing*. Vol. 15, No. 2, pp 58-69.

Bethke, A.D. 1975. *Genetic Algorithms as Function Optimizers*. Ph.D. Thesis, University of Michigan.

Bischoff, W. and Hansen, F. 1953. *Rationelles Konstruieren*, VEB-Verlag Technik, Berlin. (from Pahl and Beitz 1991)

Bock, A. 1955. *Konstruktionssystematik-die Methode der Ordnenden Gesichtspunkte, Feingeratetechnik 4*. (from Pahl and Beitz 1991)

Brown, D.C. and Chandrasekaran, B. 1986. Expert Systems for a Class of Mechanical Design Activity. *Proceedings of the First International Conference on AI Applications in Engineering*. Sriram, D. and Adey, B. Eds., Computational Mechanics Publishers, England.

Cagan, J. and Agogino, A.M. 1987. Innovative Design of Mechanical Structures from First Principles. *AI EDAM*. Vol.1, No.3, pp 169-189.

Cagan, J. and Agogino, A.M. 1988 (August). 1stPRINCE: Innovative Design from First Principles. *7th National Conference on Artificial Intelligence*. AAAI-88, Minneapolis, MN, pp. 21-26.

Chandrasekaran, B. 1988. Design: An Information Processing-Level Analysis. *Technical Report*. Laboratory for Artificial Intelligence Research, Ohio State University.

Chieng, W.H. and Hoetzel, D.A. 1987. An Interactive Hybrid System Approach to Near Optimal Design of Mechanical Components. *Engineering Optimization*. Vol. 11, pp 369-383.

Choy, J.K. and Agogino, A.M. 1986. SYMON: Automated SYMbolic MONotonicity Analysis System for Qualitative Design Optimization. *Computers in Engineering*. ASME, pp 305-310.

Codd, E.F. 1970 (June). A Relational Model for Large Shared Data Banks. *Comm. ACM*. Vol. 13, No. 6, pp. 377-387. (from Lai 1987)

Colton, J.S. and Ouellette, M.P. 1993. A Form Verification System for the Conceptual Design of Complex Mechanical Systems. *Advances in Design Automation*. Vol. 1, pp. 97-108.

Colton, J.S. et al. 1990 (September). The Requirements for an Object-Oriented Vehicle Model. *Report to General Motors Systems Engineering Center*. Georgia Institute of Technology, School of Mechanical Engineering, Atlanta, GA.

Cross, N., Naughton, J. and Walker, D. 1987. Design Method and Scientific Method. *Design: Science: Method*. Jacques, R. and Powell, J.A. (Ed.). Westbury House, Guilford, England, pp 18-27.

Crossley, E. 1980 (April 10). A Shorthand Route to Design Creativity. *Machine Design*. pp 96-100.

Dejong, K. 1975. *The Analysis and Behavior of a Class of Adaptive Systems*. PhD Thesis, University of Michigan.

Deutschman, A.D. 1975. *Machine Design: Theory and Practice*. Macmillan.

Dixon, J.R. 1988. On Research Methodology Towards a Scientific Theory of Engineering Design. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing* (AI-EDAM), Vol. 1, No. 3.

Dixon, J.R., Duffey, M.R., Irani R.K., Meunier, K.L. and Orelup, M.F. 1988. A Proposed Taxonomy of Mechanical Design Problems. *Computers in Engineering*. Vol. 1, pp 41-46.

Dixon, J.R., Howe, A., Cohen, P.R. and Simmons, M.K. 1987. Dominic I: Progress Towards Domain Independence in Design by Iterative Redesign. *Computers in Engineering*. ASME, pp 24-26.

Dyer, M.G., Flowers, M. and Hodges, J. 1984. Automating Design Invention. *Proceedings of Autofact 6 Conference*. Anaheim, CA.

Dyer, M.G., Flowers, M. and Hodges, J. 1985. EDISON: An Engineering Design Invention System Operating Naively. *Technical Report UCLA-AI-85-20.*

Edwards, K.S. and McKee, R.B. 1991. *Fundamentals of Mechanical Component Design*. McGraw-Hill, New York.

Engelmore, R.S., Morgan, A.J. and Nii, H.P. 1988. Introduction to Blackboard Systems. *Blackboard Systems*. Engelmore, R. and Morgan, T. (ed.), Addison Wesley.

Ensor, J.R. and Gabbe, J.D. 1988. Transactional Blackboards. *Blackboard Systems*. Engelmore, R. and Morgan, T. (ed.), Addison Wesley.

Erman, L.D., Hayes-Roth, F., Lesser, V.R. and Reddy, D.R. 1980. The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys*. Vol. 12, pp 213-252.

Fauvel, O.R. 1992. An Information Framework for Mechanical Design Teaching. *Proceedings of the Frontiers in Education Conference*. IEEE.

Fauvel, O.R.; Gu, P. and Norrie, D. 1993. Design Support through Information Systems. *Proceedings of the Pacific Rim Conference on Communications, Computers and Signal Processing*. IEEE. Victoria, Canada.

Finger, S. and Dixon, J.R. 1989. A Review of Research in Mechanical Design: Part I. *Research in Engineering Design*, Vol. 1, pp 51-67.

Finger, S., Fox, M.S., Prinz, F.B. and Rinderle, J.R. 1992. Concurrent Design. *Applied Artificial Intelligence*. Vol. 6, pp 257-283.

FMC Catalog LB 1200. Link-Belt Power Transmission Products

Friedman, G.J. and Leondes, C.T. 1969 (a). Constraint Theory, Part I: Fundamentals. *IEEE Transactions on Systems Science and Cybernetics*. Vol. ssc-5, No.1, pp 48-56.

Friedman, G.J. and Leondes, C.T. 1969 (b). Constraint Theory, Part II: Model Graphs and Regular Relations. *IEEE Transactions on Systems Science and Cybernetics*. Vol. ssc-5, No.2, pp 132-140.

Friedman, G.J. and Leondes, C.T. 1969 (c). Constraint Theory, Part III: Inequality and Discrete Relations. *IEEE Transactions on Systems Science and Cybernetics*. Vol. ssc-5, No.3, pp 191-199.

Gabriele, G.A. and Serrano, A.M.I. 1991. HYPERGEAR: An Object Oriented Design Program for Single Stage Gear Box Design. *Computers in Engineering*. ASME, Vol.1, pp 421-431.

GBB: User's guide. Concepts, Blackboard Technology Group, Amherst, MA.

Gero, J.S. and Coyne, R.D. 1985. Knowledge-Based Planning as a Design Paradigm. *Design Theory in Computer-Aided Design*, Proceedings of the IFIP WG 5.2 Working Conference, Tokyo, Japan, Yoshikawa, H. and Warman, E.A. (Eds.), North-Holland, Amsterdam, pp 261-295.

Gieck K. and Gieck, R. 1990. *Engineering Formulas*. 6th Edition, McGraw-Hill.

Goldberg, D.E. 1985. Optimal Initial Population Size for Binary Coded Genetic Algorithms. *TCGA Report No. 85001*. Tuscaloosa, University of Alabama, The Clearing House for Genetic Algorithms.

Goldberg, D.E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Grefenstette, J.J. 1986 (Jan./Feb.). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics*. Vol. SMC-16, No. 1.

Hansen, F. 1956. *Konstruktionssystematik*, VEB-Verlag Technik, Berlin. (from Pahl and Beitz 1991)

Hansen, F. 1965. *Konstruktionssystematik*, 2nd Ed., VEB-Verlag Technik, Berlin. (from Pahl and Beitz 1991)

Hansen, F. 1974. *Construktionswissenschaft-Grunglagen und Methoden*. Hanser, Munich. (from Pahl and Beitz 1991)

Haug, E.J. and Arora, J.S. 1979. *Applied Optimal Design: Mechanical and Structural Systems*. Wiley, New York.

Himmelblau, D.M. 1966. Decomposition of Large Scale Systems, Part 1. *Chemical Engineering Science*. Vol.21, pp 425-437.

Holland, J.H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Hoover, S.P. and Rinderle, J.R. 1989. A Synthesis Strategy for Mechanical Devices. *Research in Engineering Design*. Vol.1, No.2. pp. 87-103

Howe, A.E, Cohen, P.R., Dixon, J.R and Simmons, M.K. 1986. Dominic: A Domain-Independent Program for Mechanical Engineering Design. *Artificial Intelligence*. Vol.1, No.1, pp 23-28.

Hubka, V. 1980. *Principles of Engineering Design*. Butterworth Scientific, London.

Hundal, M.S. 1988. Methods for Systematic Development of Function Structures with Application of Computers. *Proceedings, ICED '88*. Budapest, Heurista, Zurich, pp 197-206.

Hundal, M.S. 1990. A Systematic Method for Developing Function Structures, Solutions and Concept Variants. *Mechanism and Machine Theory*. Vol. 25, No. 3, pp 243-256.

Hundal, M.S. and Byrne, J.F. 1989 (Sept. 17-2). Computer-Assisted Generation of the Requirements List in an Automated Methodical Design Procedure. *Proceedings of the 15th Design Automation Conference*. ASME, Montreal. pp 237-241

Hundal, M.S. and Byrne, J.F. 1990. Computer-Aided Generation of Function Block Diagrams in a Methodical Design Procedure. *Design Theory and Methodology DTM '90*. DE Vol. 27, ASME, New York, pp 251-257.

Hykin, D.H.W. and Lansing, L.C. 1975. Design Case Histories: Report of a Field Study of Design in the United Kingdom Engineering Industry. *Proceedings of the Institute for Mechanical Engineers*. Vol. 189, No. 23.

ICAD User's Manual. 1991. ICAD Inc., Cambrige, MA.

Ishii, K. and Barkan, P. 1987a. Design Compatibility Analysis. *Computers in Engineering*. ASME, Vol. 1, pp 95-102.

Ishii, K. and Barkan, P. 1987b. Active Constraint Reduction - A Framework for Expert Systems in Mechanical System Design. *Advances in Design Automation*. ASME, Vol.1, pp 417-424.

Jacques, R. and Powell, J.A. 1980. *Design: Science: Method*. Westbury, Guilford, England.

Jain, P. and Agogino, A.M. 1990. Theory of Design: An Optimization Perspective. *Mech. Mach. Theory*. Vol. 25, No. 3, pp 287-303.

Jenkins, W.M. 1991. Towards Structural Optimization via the Genetic Algorithm. *Computers and Structures*. Vol. 40, No. 5, pp 1321-1327.

Johnson, R.C. 1971. *Mechanical Design Synthesis*. Van Nostrand Reinhold.

Johnson, R.C. 1980. *Optimum Design of Mechanical Elements*. 2nd Ed., John Wiley and Sons.

Jones, J.C. 1972. *Design Methods-The Seeds of Human Future*. Wiley-Interscience, London.

Jones, J., Millington, M. and Ross, P. 1988. A Blackboard Shell in PROLOG. *Blackboard Systems*. Engelmore, R. and Morgan, T. (ed.), Addison Wesley.

Juster, N.P. 1985 (May). The Design Process and Design Methodologies. *Technical Report*. University of Leeds, England.

Juvinall, R.C. and Marshek, K.M. 1991. *Fundamentals of Machine Component Design*. Second Edition, Wiley.

Kannapan, S.M. and Marshek, K.M. 1990. An Algebraic and Predicate Logic Approach to Representation and Reasoning in Machine Design. *Mechanisms and Machine Theory*. Vol. 25, No. 3. pp. 335-352.

Kannapan, S.M. and Marshek, K.M. 1991. A Parametric Approach to Machine and Machine Element Design. *Int'l Journal of Mechanical Engineering Education*. Vol. 19, No. 3, pp197-211.

Kannapan, S.M. and Marshek, K.M. 1992. Engineering Design Methodologies: A New Perspective. *Intelligent Design and Manufacturing*. Kusiak, A. (Ed.), John Wiley and Sons.

Karush, W. 1939. *Minima of Functions of Several Variables with Inequalities as Side-Conditions*. Master's Thesis. Dept. of Mathematica, University of Chicago, IL. (from Kuhn and Tucker 1950)

Kitzmiller, C. and Jagannathan, V. 1989. Design in a Distributed Blackboard Framework. *Intelligent CAD I*. Yoshikawa, H. and Gossard, D. (Ed.), North-Holland.

Kogan, J. 1985. *Lifting and Conveying Machinery*. Iowa State University Press.

Koller, R. 1973. *Ein Algorithmisch-Physikalisch Orientierte Konstructionsmethodik*. Z. VDI 115. (from Pahl and Beitz 1991)

Koller, R. 1976. *Konstructionmethode fur den Maschinen: Gerate und Apparatebau.* Springer-Verlag. (from Pahl and Beitz 1991)

Koller, R. 1985. *Konstruktionslehre fur den Maschinenbau.* 2nd Ed., Springer Berlin. (from Hundal 1990)

Kota, S. and Lee, C.L. 1990. A Functional Framework for Hydraulic Systems Design Using Abstraction/Decomposition Hierarchies. *Computers in Engineering.* ASME, New York.

Krumhauer, P. 1974. *Rechnerunterstutzung fur die Konzeptphase der Konstruktion.* Dissertation, TU D 83, Berlin Technical University. (from Pahl and Beitz 1991)

Kuhn, H.W. 1976. Nonlinear Programming: A Historical View. Cottle, R. and Lemke, C. (eds.). *SIAM-AMS Proceedings.* pp 1-26.

Kuhn, H.W., and Tucker, A.W. 1950. Nonlinear Programming. Neyman, J. (ed.), *Proceeding of the Second Berkeley Symposium on Mathematical Statistics and Probability.* University of California Press, Berkeley, CA, pp 481-492.

Lai, K. 1987a. An Extended Relational Database for Conceptual Analysis of Mechanical Design. *Advances in Design Automation.* Vol. 1, pp. 391-398.

Lai, K. 1987b (May). Mechanical Design Simplification Using Function Description Language. *Proceedings of the 15th North American Manufacturing Research Conference.* pp. 615-620.

Lai, K. and Wison, W.R.D. 1987 (August). FDL-A Language for Function Analysis and

Rationalization in Mechanical Design. *Computers in Engineering*. ASME.

Laird, J.E., Newell, A. and Rosenbloom, P.S. 1989. SOAR: An Architecture for General Intelligence. AI Journal, Vol. 33, No. 1, pp 1-64, Sept. 1987

Langrana, N.A., Mitchell, T.M. and Ramchandran, N. 1986. Progress Towards a Knowledge-Based Aid For Mechanical Design. *Computers in Engineering*. ASME, pp 45-61.

Li, H.L. and Papalambros, P. 1985. A Production System for the Use of Global Optimization Knowledge. *Journal of Mechanisms, Transmissions and Automation in Design*. Vol.107, pp 277-284.

Leeson Canada Stock Catalog No. 1050C. 1993 (8/30)

Maher, M.L. 1985. HI-RISE and Beyond: Directions for Expert Systems in Design. *Computer-Aided Design*. Vol. 17, pp. 420-427.

Maher, M.L. 1987. A Knowledge-Based Approach to Preliminary Design Synthesis. *Engineering Design Research Center Paper EDRC-12-14-87*. Carnegie Mellon University.

Marples, D.L. 1961. The Decisions of Engineering Design. *IRE Transactions on Engineering Management*. Vol. EM8, pp 55-70.

Matsuta, H. and Uno, S. 1980. Applications of Advanced Integrated Designer's Activity Support System. *Man-Machine Communication in CAD/CAM*. Sata, T. and Warman, E.A. (Ed.), North-Holland.

Mehta, S.I. and Korde, U.P. 1988. An Expert System to Choose the Right Optimization Strategy. *Computers in Engineering*. ASME, pp 483-487.

Michelena, N.F. and Agogino, A.M. 1988. Multiobjective Hydraulic Cylinder Design. *Journal of Mechanisms, Transmissions and Automation in Design*. Vol.110, pp 81-87.

MICRO SWITCH Sensors and Manual Controls Catalog, Issue 8

Mistree, F. , Hughes, O.F. and Phuoc, H.B. 1981. An Optimization Method for the Design of Large, Highly Constrained Complex Systems. *Engineering Optimization*. Vol.5, pp 179-197.

Motion Control Catalog, Servosystems Inc., 1995

Newsome, S.L. and Spillers, W.R. 1989. Tools for Expert Designers: Supporting Conceptual Design. *Design Theory 88: Proceedings of the 1988 NSF Workshop on Design Theory and Methodology*. Springer-Verlag, New York.

Nicklaus, D.J., Tong, S.S. and Russo, C.J. 1987. Engenious: A Knowledge Directed Computer Aided Design Shell. *Proceedings of the 3rd Conference on AI Applications*. IEEE Computer Society.

Nii, P. 1986 (Summer). Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures, Part I. *The AI Magazine*. pp 38-53.

Oberg, E. and Jones, F.D. 1971. *Machinery's Handbook*. 19th Edition, Industrial Press Inc..

Orelup, M.F., Dixon, J.R and Simmons, M.K. 1988 (Aug. 23-26). Dominic II: Meta-Level Control in Iterative Redesign. *Proceedings of the AAAI 7th National Conference on AI*, St. Paul, MN.

Ouellette, M.P. 1992. Form Verification for the Conceptual Design of Complex Mechanical Systems. *M.S. Thesis*. Georgia Tech.

Pahl, G. and Beitz, W. 1991. *Engineering Design: A Systematic Approach*. Wallace, K. (Ed.), Springer-Verlag.

Papalambros, P.Y. 1987. Integration of knowledge Forms in Design Optimization. *Proceedings of the NSF Workshop on the Design Process*. Oakland, CA, Feb. 8-10, pp 417-438.

Papalambros, P.Y. and Li, H.L. 1983. Notes on the Operational Utility of Monotonicity in Optimization. *Journal of Mechanisms, Transmissions and Automation in Design*. Vol.105, No.2, pp 174-181.

Papalambros, P.Y. and Wilde, D.J. 1979. Global Non-Iterative Design Optimization Using Monotonicity Analysis. *Journal of Mechanical Design*. Vol.101, No.3, pp 645-649.

Papalambros, P.Y. and Wilde, D.J. 1988. *Principles of Optimal Design*. Cambridge University Press, Cambridge, England.

Parker Actuator Products Catalog 0106-1, Parker Fluidpower Inc.

Parkinson, A.R., Balling, R.J. and Free, J.C. 1984. OPTDES.BYU: A software System for Optimal Engineering Design. *Computers in Engineering*. ASME, pp 429-434.

Quayle, J. P. 1985 (Ed.). Kempe's Engineers Year-Book., Morgan-Grampian Publishing.

Ramachandran, N., Shah, A. and Langrana, N. 1988 (January). Expert System Approach in Design of Mechanical Components. *Technical Report CAIP-TR-058*, Rutgers-State University of New Jersey.

Ramachandran, N., Langrana, N. and Steinberg, L. 1990. Initial Design Strategies for Iterative Design. *Design Theory and Methodology*. ASME, Vol.27, pp 315-322.

Rao, M., Wang, Q. and Cha, J. 1992. *Integrated Distributed Intelligent Systems in Manufacturing*. Chapman and Hall.

Reklaitis, G.V., Ravindran, A. and Ragsdell, K.M. 1983. *Engineering Design Optimization*. Wiley, New York.

Rich, E. 1991. *Artificial Intelligence*. McGraw-Hill.

Rinderle, J.R. and Suh, N.P. 1982. Measures of Functional Coupling In Design. *Journal of Engineering for Industry*. ASME, Vol.104, pp 383-388.

Rinderle, J.R. and Watton, J.D. 1987. Automatic Identification of Critical Design Relationships. *Proceedings, International Conference on Engineering Design*. Cambridge, MA.

Rodenacker, W.G. 1984. *Methodisches Konstruieren*. 3rd Ed., Springer-Verlag. (from Hundal and byrne 1990)

Rodenacker, W.G. 1976. Methodisches Konstruieren. *Konstruktionsbucher*. Vol. 27, Springer-Verlag. (from Pahl and Beitz 1991)

Rodenacker, W.G. and Claussen, U. 1974. *Regeln des Methodischen Konstruierens*, Mainz: Krausskopf. (from Pahl and Beitz 1991)

Rosen, D., Erdman, A. and Riley, D. 1987. A General Design Knowledge-based System Shell, With Application to Dwell Mechanism Design. *Computers in Engineering*. ASME, pp. 29-36.

Roth, K. 1982. *Konstruieren mit Konstruktionsanlagen*. Springer-Verlag, Berlin. (from Pahl and Beitz 1991)

Roth, K., Frank, H.J. and Simonek, R. 1972. Die Allgemeine Funktionsstructur, Ein Wesentliches Hillfsmittel zum Methodischen Konstruieren. *Konstruction*. 24, pp 277-282. (from Pahl and Beitz 1991)

Sambura, A.S. and Gero, J.S. 1982. Framework for a Computer Integrated Design Environment. *CAD System Framework*. Bo. K. and Lillehagen, F.M. (Ed.), North-Holland.

Schon, D.A. 1988 (July). Designing: Rules, Types, and Worlds. *Design Studies*, Vol. 9, No. 3.

Seering, W.P. 1985 (April). Who Said Robots Should Work Like People? *Technology Review*. pp 58-67

Serrano, D. 1984. *MATHPAK: An Interactive Preliminary Design Package*. M.S. Thesis, MIT.

Serrano, D. 1990. Managing Constraints in Concurrent Design: First Steps. *Computers in Engineering*. ASME, pp 159-164.

Serrano, D. and Gossard, D. 1987. Constraint Management in Conceptual Design. *Knowledge Based Expert Systems in Engineering: Planning and Design*. Sriram, D. and Adey, R.A. Ed., Computational Mechanics Publications, England, pp 211-224.

Serrano, D. and Gossard, D. 1988. Constraint Management in MCAE. *Artificial Intelligence in Engineering Design*. Gero, J.S. Ed., Elsevier Science Publishers, Amsterdam, pp 217-240.

Shah, A., Ramachandran, N. and Langrana, N. 1987 (November). Knowledge Based Design of Primitive Mechanical Components. *Technical Report CAIP-TR-052*. Centre for Computer Aids for Productivity, Rutgers University.

Shigley, J.E. 1986. *Mechanical Engineering Design*. First Metric Edition, McGraw-Hill.

Shigley, J.E. and Mischke, C.R. 1986. *Standard Handbook of Machine Design*. McGraw-Hill.

Shigley, J. E. and Mischke, C. R. 1989. *Mechanical Engineering Design*. 5th Ed. McGraw-Hill.

Siddall, J.N. 1982. *Optimal Engineering Design*. Marcel Dekker.

Simon, H.A. 1969. *The Sciences of the Artificial*, MIT Press, Cambridge, MA.

Soylemez, S. and Seider, W.D. 1973. A New Technique for Precedence-Ordering Chemical Process Equation Sets. *AIChE Journal*. Vol.19, No.5, pp 934-942.

Sridhar, N., Agrawal, R., Kinzel, G.L. 1991. An Automatic Approach to Handling Inequality Constraints in an Interactive Design Environment. *Advances in Design Automation*. Vol. 1, pp 219-223.

Steinberg, L.I. 1987 (July). Design as Refinement Plus Constraint Propagation: The VEXED Experience. *Proceedings of the Sixth National Conference on Artificial Intelligence*. AAAI, Seattle, WA.

Steward, D.V. 1981. *Systems Analysis and Management*. Princeton.

Strakosch, G. R. 1983. *Elevators and Escalators*. 2nd Ed., John Wiley and Sons.

Sturges, R.H. 1992. A Computational Model for Conceptual Design Based on Function Logic. *Artificial Intelligence in Design*. Gero, J.S. (Ed.), pp 757-772.

Sturges, R.H. and Kilani, M.I. 1990. A Function Logic and Allocation Design Environment. Proceedings for ESD Fourth Annual Expert Systems Conference and Exposition, Detroit, MI.

Suh, N.P. 1990. *The Principles of Design*. Oxford University Press.

Taguchi, G. 1978. Off-Line and On-Line Quality Control Systems. *Int'l Conference on Quality Control*, Tokyo.

Taguchi, G. 1987. *System of Experimental Design*. Vols. 1 and 2, UNIPUB-Kraus International Publications.

Taguchi, G. and Wu, Y. 1980. Introduction to Off-Line Quality Control. *Central Japan Quality Control Association*.

Tebay, R., Atherton, J. and Wearne, S.H. 1984. Mechanical Engineering Design Decisions: Instances of Practice Compared with Theory. *Proceedings of the Institute for Mechanical Engineers*. Vol. 198B, No. 6.

Terry, A. 1988. Using Explicit Strategic Knowledge to Control Expert Systems. *Blackboard Systems*. Engelmore, R. and Morgan, T. (ed.), Addison Wesley.

Thompson, T.R., Riley, D.R. and Erdman, A. 1985. An Expert System Approach to Type Synthesis of Mechanisms. *Computers in Engineering*. ASME.

TK Solver Plus, Introduction. 1988. Universal Technical Systems, Inc., Rockford, Illinois.

Ullman, D.G. 1989. A Taxonomy of Mechanical Design. *Design Theory and Methodology DTM '89*. ASME, DE-Vol. 17, pp 23-36.

Ullman, D.G. and Dietterich, T. A. 1986. Mechanical Design Methodology. *Computers in Engineering*, ASME, pp 173-180.

Ullman, D.G., Stauffer, L.A. and Dietterich, T.G. 1987. Preliminary Results of an Experimental Study of the Mechanical Design Process. *Proceedings of the NSF Workshop on Design Process*, Waldron, M.B. (Ed.), pp 145-188, Feb. 8-10.

Ulrich, K.T. and Seering, W.P. 1988a (August). Function Sharing in Mechanical Design. *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*. St. Paul, MN. pp 342-347.

Ulrich, K.T. and Seering, W.P. 1988b (October). Computation and Pre-Parametric Design. *MIT Artificial Intelligence Laboratory Technical Report 1043*.

Ulrich, K.T. and Seering, W.P. 1989. Synthesis of Schematic Descriptions in Mechanical Design. *Research in Engineering Design*. Vol. 1, No. 1, pp 3-18.

U.S. Air Force. 1981 (June). *Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II, Volume IV- Functional Modelling Manual (IDEF0)*. Air Force Materials Laboratory, Wright-Patterson AFB, Ohio 45433, AFWAL-TR-81-4023.

Waldron, M.B., Jelinek, W., Owen, D. and Waldron, K.J. 1987. A Study of Visual Recall Differences Between Expert and Naive Mechanical Designers. *Int'l Conference on Engineering Design*, ICED, Cambridge, MA, pp 86-93.

Wallace, K.N. and Hale, C. 1987. Detailed Analysis of an Engineering Design Project. *Proceedings of the Int'l Conference on Engineering Design, ICED*, Cambridge, MA, pp 94-101.

Wilde, D.J. 1975. Monotonicity and Dominance in Optimal Hydraulic Cylinder Design. *Journal of Engineering for Industry*. Vol.97, pp 1310-1314.

Wilde, D.J. 1978. *Globally Optimal Design*. Wiley, New York.

Wilde, D.J. 1986. A Maximal Activity Principle for Eliminating Overconstrained

Optimization Cases. *Journal of Mechanisms, Transmissions and Automation in Design*. Vol.108, pp 312-314.

Yin, F.Y. 1992. *An Information Structure for Mechanical Design*. Master's Thesis. University of Calgary, Calgary, Canada.

Yoshikawa, H. 1982. CAD Framework Guided by General Design Theory. *CAD System Framework*. Bo. K. and Lillehagen, F.M. (Ed.), North-Holland.

Zarefar, H., Lawley, T.J. and Etesami, F. 1986. PAGES: A Parallel Axis Gear Drive Expert System. *Computers in Engineering*. ASME, pp 145-147.

Zhang, Z. and Rice, S.L. 1989. An Expert System for Conceptual Mechanical Design. *Computers in Engineering*. ASME, Vol. 1, pp. 281-285.

GLOSSARY

The following is an alphabetical glossary of the major terms that recur throughout this work. The definition of each term is followed by a reference to where it is first articulated in the text.

Blackboard Model:	A general model of problem solving based on the cooperation of multiple domain experts that can simultaneously access/affect a common representation of a design problem (Sec. 4.3).
Chromosome:	A (binary) string of specified length representing the encoded values of a certain number of design variables of a search/optimization problem (Sec. 5.2.5.1).
Conceptual Design:	The middle stage of the mechanical design process wherein the standard functional description of a product is transformed to a generic structural description and thus a configuration of components is specified which will carry out the specified function(s) (Sec. 1.4).
Concurrent Design:	The process of mechanical design in which requirements and implications of various stages of a product's lifecycle are considered concurrently rather than sequentially (Sec. 1.4).
Crossover:	Random, partial swapping of two mating chromosomes for the purpose of producing new ones (Sec. 5.2.5.1).
(Element) Knowledge Cell:	A component-specific database unit containing the component's design equations and constraints as well as other design-related information (Sec. 3.1).
(Element) Knowledge pool:	Combination of all the constraints applicable to a component plus any other given information (e.g. initial specifications) (Sec. 3.2.3).

Evaluation Knowledge-Source:	The third of the four "generating" knowledge-sources of the Conceptual Designer's blackboard architecture; in charge of determining the contribution of the explored candidate components to the satisfaction of the overall requirements of a problem (Sec. 4.4.2).
Expert Module/ Knowledge Source:	Each of several domain-expert subroutines in a blackboard model, each watching and possibly modifying partial designs as they develop (Sec. 4.3).
Exploration Knowledge-Source:	The second of the four "generating" knowledge-sources of the Conceptual Designer's blackboard architecture; in charge of evaluating and solving the constraint sets of the candidate components (Sec. 4.4.2).
External Constraints:	Qualitative/quantitative constraints included in the initial requirements of a design problem (Sec. 3.2.1).
Fitness:	The value of the objective (fitness) function for the variables represented by a particular chromosome, representing the goodness of that chromosome (Sec. 5.2.5.1).
Functional Design:	The early stage of the mechanical design process wherein the overall, abstract function of a product is refined and expressed in terms of standard functions (Sec. 1.4).
Function Block Diagram (FBD):	A structured graphical representation of the function(s) of a mechanical system consisting of a set of nodes (functions) and arcs (relations) (Sec. 1.9).

Genetic Algorithms:	A general purpose search/optimization method based on the principles of natural genetics and survival of the fittest (Sec. 5.2.5.1).
Incidence Matrix (of an Equation Set):	The incidence matrix $[a_{ij}]$ of a set of p equations in q variables is a $p \times q$ matrix in which $a_{ij} = 1$ if variable j appears in equation i, and $a_{ij} = 0$ otherwise (Sec. 5.2.2).
Instantiation:	The process of finding a number of feasible instances of a selected component by solving its constraint set for given specifications (Sec. 3.1).
Internal Constraints:	Equality/inequality constraints initially contained in a component's knowledge-cell, including its design equations (Sec. 3.2.1).
Mutation:	Occasional changing of a randomly selected bit in a chromosome from 0 to 1 or vice versa (Sec. 5.2.5.1).
Nomination Knowledge-Source:	The first of the four "generating" knowledge-sources of the Conceptual Designer's blackboard architecture; in charge of finding qualified candidate components (i.e. the ones that can carry out one or more of the desired, specified functions) (Sec. 4.4.2).
Parametric Design:	The last stage of the mechanical design process wherein the values of the function- and form-defining parameters of every component in the above configuration are determined (Sec. 1.4).
Partitioning (the Incidence Matrix of an Equation Set):	Rearranging the incidence matrix into a block-triangular form so that each block on the main diagonal would represent an irreducible subset of equations to be solved simultaneously (Sec. 5.2.5.4).
Population Size:	The (fixed) number of chromosomes processed in each iteration of the GA (generation) (Sec. 5.2.5.1).

Reproduction:	The process of producing new, generally better, chromosomes by applying genetic operators to existing ones (Sec. 5.2.5.1).
Standard Elemental Functions (SEFs):	A set of standard, supposedly universal mechanical functions used as a vocabulary to express the higher-level functions of mechanical systems (Sec. 1.9.1).
System Constraints:	Constraints generated by the design program to reflect the implications of selecting a particular component (Sec. 3.2.1).
Variable Matching:	Setting up a one-to-one assignment between each equation in an equation set and a variable in that equation (Sec. 5.2.2).
Verification Knowledge-Source:	The fourth of the four "generating" knowledge-sources of the Conceptual Designer's blackboard architecture; in charge of verifying the validity of each generated (partial) design (Sec. 4.4.2).

APPENDIX A

A DIRECTORY OF THE *STANDARD ELEMENTAL FUNCTIONS*

The following is a directory of the *Standard Elemental Functions (SEFs)* introduced in section 1.9.1. SEFs form the vocabulary wherein the function(s) of a mechanical system, or any component thereof, is formally described. As discussed in chapter 1, these *functional descriptions* are systematically generated through the Functional Design of a device.

To generate this directory, we have introduced eight basic functions, of which five (Supply, Keep, Damp/Dissipate, Sense and split) primarily deal with single entities whereas the other three (Coalesce, Compare and join) deal with multiple entities. Each of these basic functions has then been applied to the three main quantities: material, energy and signal, and sub-classes have been formed accordingly. Some of the finer sub-classes are not shown here due to space limitations. For instance, the sub-class "Change-Form-Material-Solid" can be broken up into "Crush", "Forge", "Extrude", "Machine", etc.

Whereas the following classification of the mechanical functions is by no means unique, it results in a fairly standard set of elemental functions. Other classifications may start with a different set of basic functions and take different appraoches to breaking them up, but they will one way or another result in similar elemental functions. In this sense, the presented directory can be considered universal and be used to represent various mechanical systems.

Supply	Material	Solid		
		Liquid		
		Gas		
	Energy	Mechanical	Translational	
		Mechanical	Reciprocal	
		Mechanical	Rotational	
		Mechanical	Vibrational	
		Electrical		
		Chemical		
		Magnetic		
		Light		
		Sound		
	Signal			
Keep	Store	Material	Solid	
			Liquid	
			Gas	
		Energy	Mechanical	Translational
			Mechanical	Reciprocal
			Mechanical	Rotational
			Mechanical	Vibrational
			Electrical	
			Chemical	
			Magnetic	
			Light	
			Sound	
	Sustain	Energy (Mechanical)	Static	
			Dynamic	Translational
				Rotational
Damp/ Dissipate	Energy	Mechanical	Translational	
		Mechanical	Reciprocal	
		Mechanical	Rotational	
		Mechanical	Vibrational	
		Electrical		
		Chemical		
		Magnetic		
		Thermal		
		Light		
		Sound		

Change	Form	Material(Solid)		
		Signal	Integrate Differentiate	
	Magnitude	Energy	Displacemen	Linear Rotational
			Velocity	Linear Rotational
			Acceleration	Linear Rotational
			Force Torque Pressure Stress Density Friction Electrical Magnetic Thermal Light Sound	
		Signal		
	Direction	Displacement	Linear Rotational	
		Velocity	Linear Rotational	
		Acceleration	Linear Rotational	
		Force Torque		

Transport/ Transmit	Material	Solid Liquid Gas		
	Energy	Mechanical	Displacement	Linear Rotational
			Velocity	Linear Rotational
			Acceleration	Linear Rotational
			Force	
			Torque	
			Pressure	
			Stress	
			Density	
		Electrical Magnetic Thermal Light Sound		
	Signal			
Sense	Detect	Material	Solid Liquid Gas	
		Energy	Mechanical Electrical Chemical Magnetic Thermal Light Sound	
		Signal		

	Measure	Quality	Material Energy Signal	Various Characteristics
		Quantity		
	Display	Signal		
	Mark	Material	Solid Liquid Gas	
Split	Branch	Material	Solid Liquid Gas	
		Energy	Mechanical Electrical Magnetic Thermal Light Sound	
		Signal		
	Isolate	Material	Solid Liquid Gas	
		Energy	Mechanical Electrical Magnetic Thermal Light Sound	

	Separate	Material	Solid Liquid Gas	
		Signal		
	Switch	Material	Solid Liquid Gas	
		Energy	Mechanical Electrical Magnetic Thermal Light Sound	
		Signal		
	Valve	Material	Solid Liquid Gas	
		Energy	On-Off	Mechanical Electrical Magnetic Thermal Light Sound
			Continuous	
		Signal	On-Off	
			Continuous	

Coalesce	Join	Material (Solid)		
	Add	Material	Solid Liquid Gas	
		Energy	Mechanical Electrical Chemical Magnetic Thermal Light Sound	
		Signal	Add Subtract Multiply Divide And Or	
	Mix	Material	Solid Liquid Gas	
	Compare/ sort	Quality	Material Energy Signal	Various Characteristics
		Quantity		

APPENDIX B

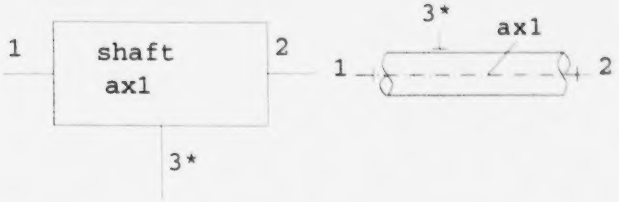
STRUCTURAL REPRESENTATION OF MECHANICAL SYSTEMS

In Section 1.9 we discussed our method of representing the functional behavior of a device. In this appendix we provide an overview of the method we use to represent the physical structure of a device. Note that the method introduced here is intended to exclusively describe the *structure* of the artifacts and not their behavior or function. As mentioned in Chapter 4, throughout the design process the *conceptual designer* continually displays the structure of partial designs on the structural-representation panel of the system's blackboard. It was also mentioned that the final products of the design process, i.e. the completed designs, will be graphically presented to the user.

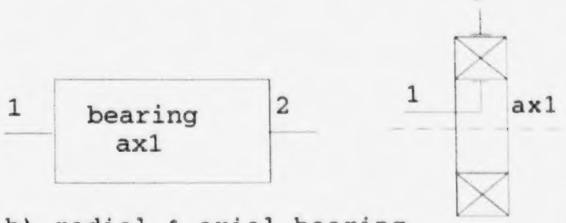
We formally represent mechanical devices as well as their components using a set of algebraic and predicate logic statements. The presented method can be considered a simplified version of the *representation scheme* proposed by Kannapan and Marshek (Kannapan and Marshek 1990). The major difference between our method and the one reported in Kannapan and Marshek's work is that our representation of a device does *not* encompass the *function* or *behavior* of the device. This is because we have chosen, for the sake of clarity and convenience, to use a different method, namely the Function Block Diagrams (Chapter 1), for our functional representation purposes. This choice of ours has been motivated by the fact that, compared to Kannapan and Marshek's proposed scheme, FBDs would provide us with more flexibility and accuracy when it comes to such aspects of a design as the complex interrelationships among the individual functions of the device.

We represent *objects* (systems as well as their individual components) both graphically and symbolically. Either representation provides information about the object's constituent *elements* as well as the *configuration* and *interconnections* of these elements. Graphically, each component is represented as a box with a number of "ports" connected to its perimeter. Contained in each box is the name of the component as well as a number (≥ 1) of component *attributes.* Attributes are significant structural features of a component which contribute to the characterization of its function. Axis of a shaft and surface of a friction clutch are examples of attributes. Component boxes are interconnected through their ports to form systems. It is through these ports that various entities (material, energy and signal) are transfered from one object to another.

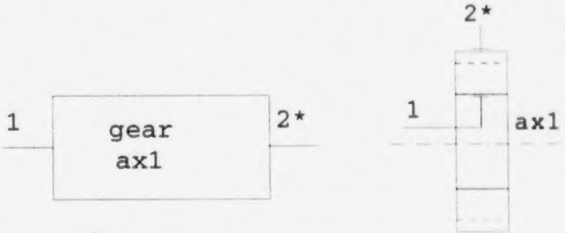
Figure B-1 shows the graphical representation of four single components (shaft, bearing, external spur gear and machine frame). Consider, for instance, Figure B-1(a). The information inside the box tells us that the component is a shaft with the axis (ax1). Three ports are shown connected to the box. Ports 1 and 2 indicate that other objects can be connected to either end of the shaft. Port 3 refers to the possibility of other objects being mounted on the shaft. An asterisk (*) next to the port number indicates that the port may be replicated an indefinite number of times in an instance of the shaft, i.e. an indefinite number of objects may be connected to the shaft through instances of this port (numbered as 3.1, 3.2. etc.). Also, an integer appearing in brackets next to a port number indicates the minimum number of times a particular port should be replicated.



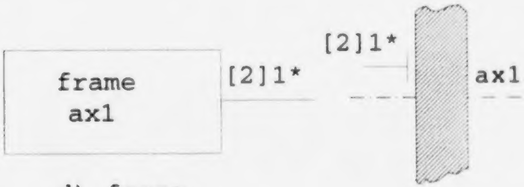
a) solid shaft



b) radial & axial bearing



c) spur gear



d) frame

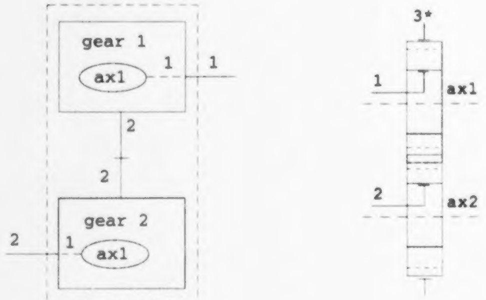
Figure B-1: Graphical and schematic representation of four typical components

Figure B-2 shows the graphical representation of two subsystems "gear_pair" and "shaft_support". In each case the dashed-box represents the subsystem boundaries. As could be seen in Figure B-2(a), each of the two mating gears have two port instances: one representing the gear's hub and the other representing the external meshed contact with the other gears. Note that the second port in each gear originally has a (*) associated with it [Figure B-1(c)], meaning that the gear can be externally engaged with an indefinite number of other gears. However, since in this case there is only one external engagement, the second port has been instantiated just once.

We have chosen port 1 of gear 1 and port 1 of gear 2 to represent the two gear hubs. These two ports are therefore free for later connections. Note that the two ports have been numbered differently inside and outside the subsystem boundaries. Whereas they are both "port 1" internally, they become "port 1" and "port 2" at the subsystem boundaries. The other two ports (port 2 of gear 1 and port 2 of gear 2) are connected together to represent the engagement of the two gears.

In Figure B-2(b) ports 1 and 2 of shaft 1 are free for future connections whereas the first instance of its third port (i.e. port 3.1) is connected to port 1 of the bearing, indicating that the bearing is mounted on the shaft. The other instance(s) of the shaft's port 3 (i.e. ports 3.2, 3.3, etc. as implied by the asterisk) remain open, meaning that other components can be mounted on the same shaft. Meanwhile, port 2 (external perimeter) of the bearing is connected to port 1.1 of the frame, implying that the bearing is externally supported by the frame. The other two (or more, as implied by the asterisk) ports of the frame (i.e. ports 1.2, 1.3,

a) gear pair



b) shaft support

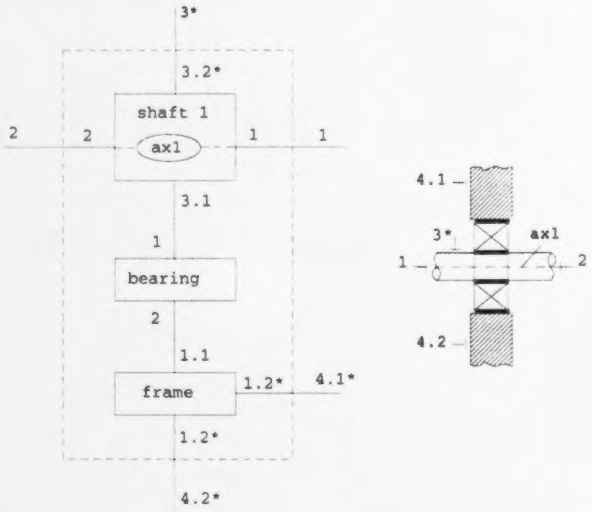


Figure B-2: Graphical representation of two subsystems

1.4, etc.) are reserved for other components to be supported by the frame.

Graphical representation is only used to present to the user the solution(s) generated by the design system, and will not be utilized by the system in its design activities.

Compared to graphical representation, symbolic representation of devices plays a more prominent role in the design process, as the majority of the design information posted to the system's blackboard by various contributing modules is reflected in this type of design representation. This information typically includes the current status of (partial) design(s), functional requirements to be further satisfied, internal constraints stemming from interactions among the components, and some of the quantitative information required to carry out the rest of the design.

Symbolic representation of an object comprises a set of "algebraic expressions" and "predicate statements". Algebraic expressions generally tell us which component(s) a device is composed of and how these components are connected to one another, whereas the predicate statements provide quantitative information as well as that part of the qualitative information which is normally not included in the graphical representation of the device. The algebraic part of a component's symbolic representation typically has the following format.

(component name; port names; attribute name: attribute type)

Where parentheses delimit representations, semicolons separate sets of specifications and commas separate members of a set (if more than one). For example, the symbolic (algebraic) representations of components shown in Figure B-1 are as follows respectively.

(shaft; 1, 2, 3; ax1: axis-type)*

(bearing; 1, 2; ax1: axis-type)

(gear; 1, 2; ax1: axis-type)*

(frame; [2]1; ax1: axis-type)*

Similarly, the algebraic part of a system/subsystem's symbolic representation typically has the following format.

(system name; free port names; component names; PCons)

Where "free ports" of a system are the few component ports that remain untied internally and are reserved for connection to the free ports of other systems. The *PCons* (Port Connections) part provides the information about interconnections among the system's components. PCons is the set of pairs (two-tuples) of component-port names where each pair represents a single connection between two components (one port from each). As an example, the algebraic representations of the sub-systems of Figure B-2 are presented below. Note that in this example we have assigned a code of the form (Sn: n = 1, 2, ...) to each system to distinguish between similar elements of different systems.

S1:= (gear_pair; gear_1: 1, gear_2: 1; gear_1, gear_2; PCons [S1])

PCons[S1]:= {(gear_1: 2, gear_2: 2)}

In the above example, *gear_pair* is the name of the subsystem, *gear_1* and *gear_2* are the names of its components and *gear_1: 1* (port 1 of gear_1) and *gear_2: 1* (port 1 of gear_2) are the free ports of the subsystem. Also, the only member of the set of port connections of the subsystem (PCons[S1]) represens the tie between port 2 of gear_1 and port 2 of gear_2 [Figure B-2(a)].

Similarly, the subsystem of Figure B-2(b) is represented as follows.

S2:= (shaft_support; shaft: 1, shaft: 2, shaft: 3.2, frame: 1.2*, frame: 1.3*;*
shaft, bearing, frame; PCons[S2])

PCons [S2]:= {(shaft: 3.1, bearing: 1), (bearing: 2, frame: 1.1)}

Here again we have a minimum of five free ports, namely ports 1 and 2 of the shaft, a number (≥ 1) of port 3's of the shaft where the latter refers to the indefinite number of components that can be mounted on the shaft, and a number (≥ 2) of port 1's of the frame. We also have three components, namely the shaft, the bearing and the frame, and two port connections as illustrated in Figure B-2(b).

With this we conclude our discussion of our structural representation method. There are some fine details of the method which we did not, and are not going to, discuss here. This is because in the current research, "representation" is regarded only as a means for communucation and does not play a crucial role in the process of design itself.

APPENDIX C

TYPICAL SEARCH PATTERN OF THE GENETIC ALGORITHM (PARTIALLY SHOWN)

The evolution of the generations for the gear-design example from a typical run of the GA is illustrated here. In the following tables values of Penalty Function (PF) have been shown instead of the Objective Function to make it easier to study the rate of convergence. An asterisk (*) in the last column denotes a solution (PF=0).

Generation	Family	ML(mm)	NPT	AlS(MPa)	FW(mm)	PF
1	1	2.5	24	20.0	12	1.127
	2	1.5	26	50.0	14	1.202
	3	3.0	40	72.5	18	0.541
	4	1.5	30	57.5	35	1.336
	5	5.0	28	57.5	80	0.000 *
	6	16.0	30	50.0	14	16.423
	7	4.0	17	50.0	14	1.012
	8	12.0	19	20.0	70	0.787
	9	6.0	16	20.0	70	0.354
	10	6.0	20	125.0	12	0.504
	11	9.0	34	57.5	16	3.665
	12	16.0	19	125.0	12	3.676

Table contd.

Generation	Family	ML(mm)	NPT	AIS(MPa)	FW(mm)	PF
2	1	4.0	38	20.0	12	1.065
	2	1.5	24	20.0	14	1.280
	3	6.0	17	57.5	18	0.689
	4	12.0	34	57.5	45	9.786
	5	9.0	28	125.0	80	1.322
	6	2.5	40	20.0	16	0.935
	7	12.0	18	50.0	70	0.554
	8	1.5	18	50.0	55	4.027
	9	3.0	17	50.0	70	0.934
	10	6.0	15	57.5	12	0.978
	11	5.0	38	57.5	14	0.733
	12	16.0	19	125.0	12	3.675
3	1	3.0	40	50.0	12	0.899
	2	3.0	26	50.0	45	0.426
	3	6.0	16	50.0	18	0.777
	4	16.0	17	57.5	18	2.357
	5	6.0	15	72.5	80	0.000 *
	6	1.5	18	55.0	16	1.391
	7	16.0	18	20.0	55	2.758
	8	2.5	19	20.0	70	1.884
	9	1.5	16	57.5	40	2.114
	10	9.0	28	57.5	12	1.803
	11	20.0	34	57.5	14	42.691
	12	6.0	40	125.0	25	1.150

Table contd.

Generation	Family	ML(mm)	NPT	AlS(MPa)	FW(mm)	PF
5	1	4.0	38	55.0	12	0.813
	2	4.0	28	55.0	80	0.103
	3	6.0	26	72.5	60	0.000 *
	4	16.0	16	50.0	30	1.765
	5	2.5	14	72.5	55	1.185
	6	9.0	15	72.5	55	0.054
	7	6.0	21	55.0	70	0.000 *
	8	6.0	19	125.0	45	0.015
	9	1.5	16	20.0	16	1.475
	10	1.0	30	55.0	10	1.364
	11	4.0	34	57.5	18	0.567
	12	12.0	16	20.0	40	0.447
8	1	3.0	26	50.0	45	0.426
	2	6.0	34	55.0	60	0.303
	3	6.0	19	125.0	60	0.000 *
	4	6.0	14	125.0	55	0.000 *
	5	6.0	19	125.0	55	0.000 *
	6	6.0	28	57.5	70	0.010
	7	10.0	16	72.5	60	0.058
	8	8.0	19	72.5	60	0.015
	9	6.0	26	20.0	80	0.045
	10	6.0	15	72.5	45	0.031
	11	4.0	22	72.5	45	0.097
	12	4.0	21	72.5	80	0.103

Table contd.

Generation	Family	ML(mm)	NPT	AlS(MPa)	FW(mm)	PF
10	1	6.0	19	125.0	70	0.000 *
	2	6.0	15	72.5	60	0.000 *
	3	6.0	18	57.5	55	0.000 *
	4	6.0	19	57.5	60	0.000 *
	5	6.0	19	57.5	35	0.146
	6	8.0	15	72.5	60	0.014
	7	10.0	14	125.0	60	0.058
	8	10.0	15	125.0	70	0.026
	9	4.0	19	55.0	70	0.117
	10	10.0	21	72.5	80	0.397
	11	6.0	22	72.5	50	0.003
	12	4.0	16	125.0	45	0.079
14	1	2.5	14	57.5	45	1.098
	2	6.0	18	72.5	55	0.000 *
	3	4.0	24	57.5	55	0.070
	4	6.0	18	125.0	50	0.003
	5	6.0	19	125.0	55	0.000 *
	6	6.0	19	125.0	50	0.003
	7	6.0	18	125.0	35	0.064
	8	10.0	15	125.0	35	0.194
	9	6.0	17	72.5	70	0.000 *
	10	6.0	16	72.5	60	0.000 *
	11	4.0	22	125.0	45	0.000 *
	12	6.0	14	57.5	45	0.148

Table contd.

Generation	Family	ML(mm)	NPT	AIS(MPa)	FW(mm)	PF
17	1	6.0	16	72.5	45	0.017
	2	6.0	16	125.0	80	0.000 *
	3	6.0	18	72.5	50	0.003
	4	4.0	34	125.0	35	0.000 *
	5	4.0	16	125.0	60	0.040
	6	6.0	15	125.0	70	0.000 *
	7	6.0	18	72.5	70	0.000 *
	8	6.0	19	125.0	70	0.000 *
	9	5.0	19	72.5	55	0.000 *
	10	6.0	19	72.5	35	0.074
	11	6.0	19	72.5	55	0.000 *
	12	6.0	17	72.5	55	0.000 *
18	1	6.0	17	72.5	80	0.000 *
	2	6.0	16	125.0	45	0.014
	3	6.0	19	72.5	70	0.000 *
	4	4.0	34	125.0	35	0.000 *
	5	6.0	19	125.0	55	0.000 *
	6	6.0	14	125.0	50	0.002
	7	6.0	18	72.5	70	0.000 *
	8	6.0	21	125.0	70	0.000 *
	9	5.0	18	72.5	55	0.000 *
	10	6.0	15	72.5	70	0.000 *
	11	6.0	19	72.5	55	0.000 *
	12	6.0	15	72.5	55	0.000 *

APPENDIX D

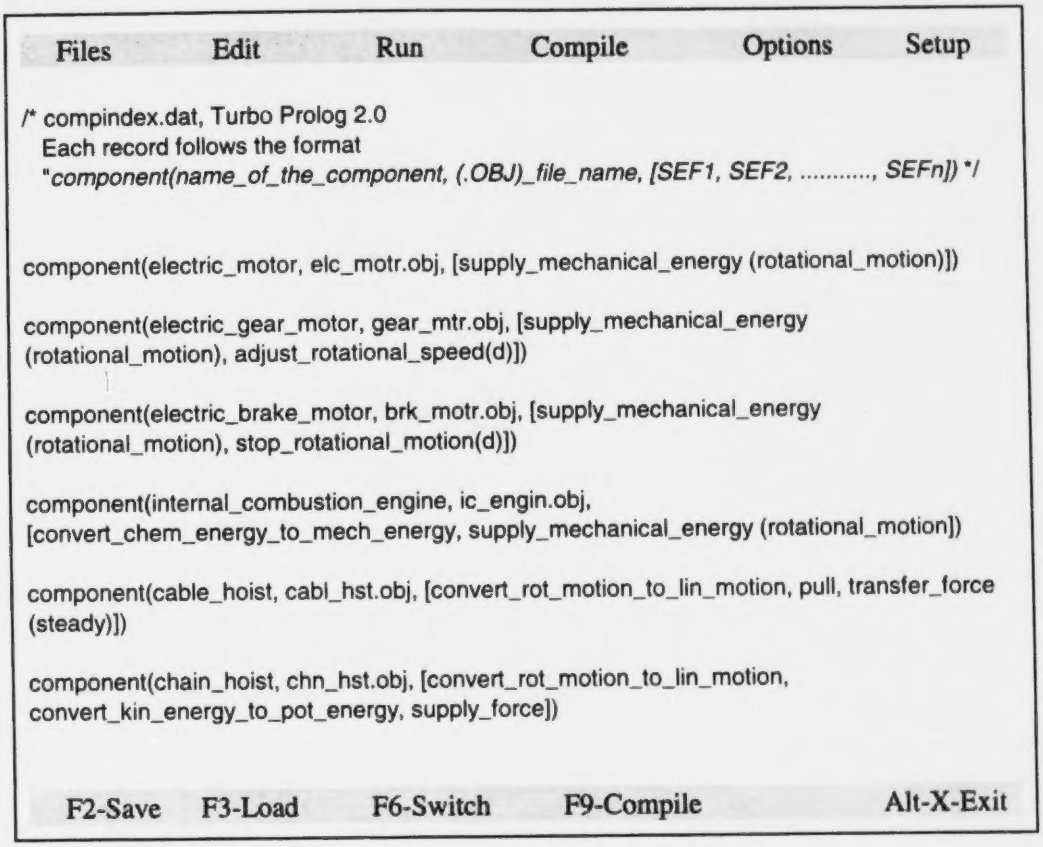
INDEX TO THE COMPONENTS-DATABASE

The following index is used to retrieve the names and the file-addresses of the components capable of performing a specified Standard Elemental Function. Given an SEF, the Conceptual Designer scans this index and spots all the components in the components-database that have the specified SEF in their list of functions. It then returns the names of those components and the file-addresses of corresponding component-cells (.OBJ files). These component-cells are later used to form knowledge-pools of the candidate components (in the form of Turbo Prolog's *projects*), which in turn will be used in the exploration of the functional behavior of the candidate components.

For brevity, only part of the index, mostly those records used in the example of chapter 6, are shown here. Each record has the following format.

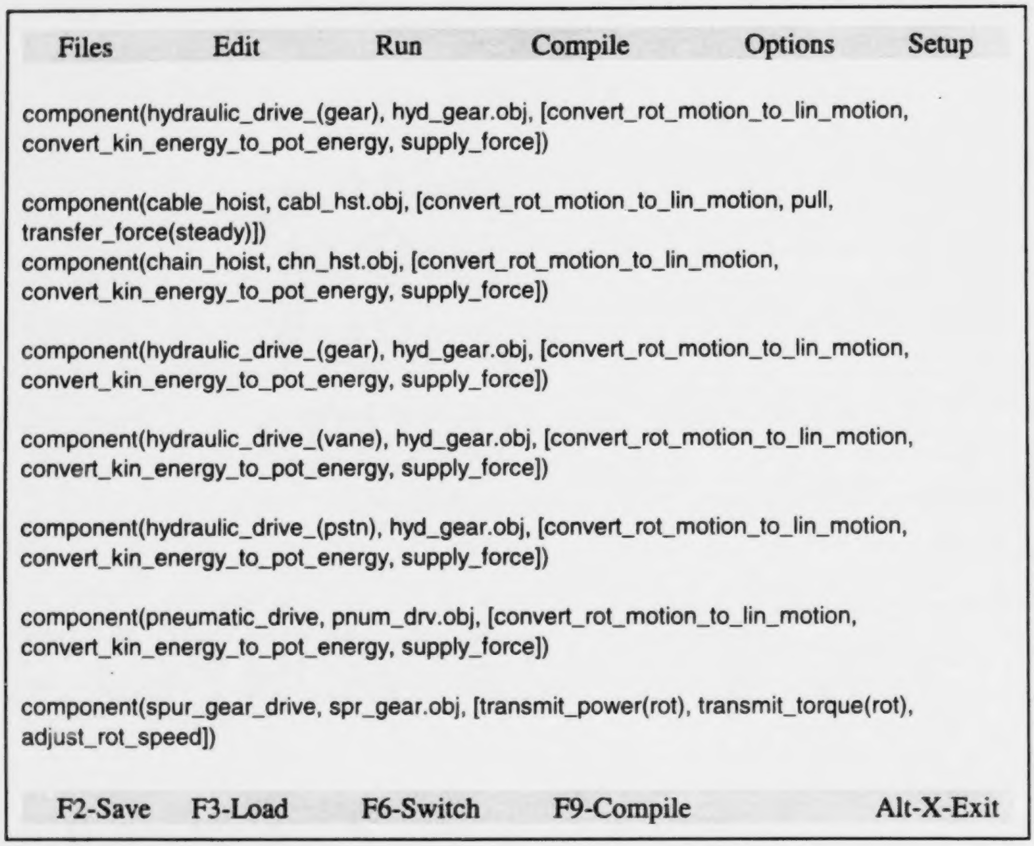
component(name_of_the_component, (.OBJ)_file_name, [SEF1, SEF2, ..., SEFn])

where *n* is the number of Standard Elemental Functions the component potentially performs. Also, the letter "d" following a function indicates that the function is *dependent on* or *coupled with* the other function(s) performed by that component. For example, the function "brake" in a brake-motor is always associated with the other function of the component, i.e. "supply rotational motion", and cannot be used independently to, say, stop the motion of another object.



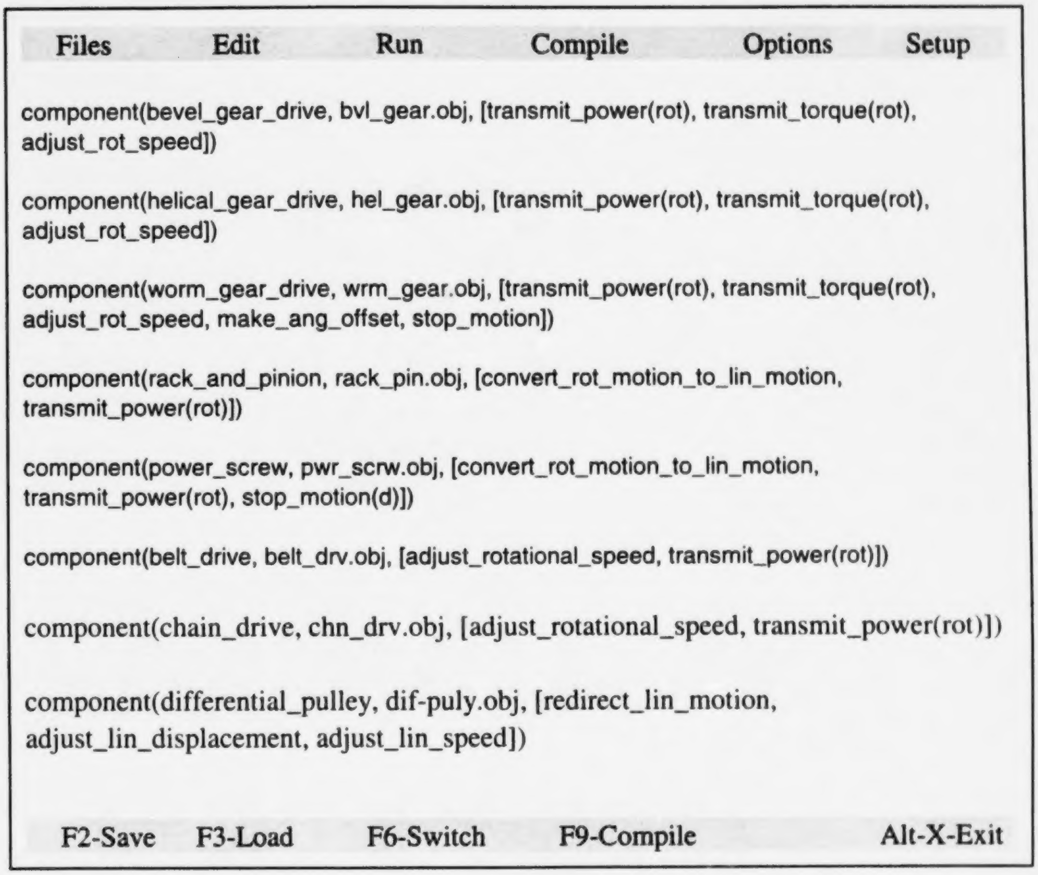
```
/* compindex.dat, Turbo Prolog 2.0
  Each record follows the format
  "component(name_of_the_component, (.OBJ)_file_name, [SEF1, SEF2, ..........., SEFn]) */


component(electric_motor, elc_motr.obj, [supply_mechanical_energy (rotational_motion)])

component(electric_gear_motor, gear_mtr.obj, [supply_mechanical_energy
(rotational_motion), adjust_rotational_speed(d)])

component(electric_brake_motor, brk_motr.obj, [supply_mechanical_energy
(rotational_motion), stop_rotational_motion(d)])

component(internal_combustion_engine, ic_engin.obj,
[convert_chem_energy_to_mech_energy, supply_mechanical_energy (rotational_motion])

component(cable_hoist, cabl_hst.obj, [convert_rot_motion_to_lin_motion, pull, transfer_force
(steady)])

component(chain_hoist, chn_hst.obj, [convert_rot_motion_to_lin_motion,
convert_kin_energy_to_pot_energy, supply_force])
```

Figure D-1: Index to the components-database



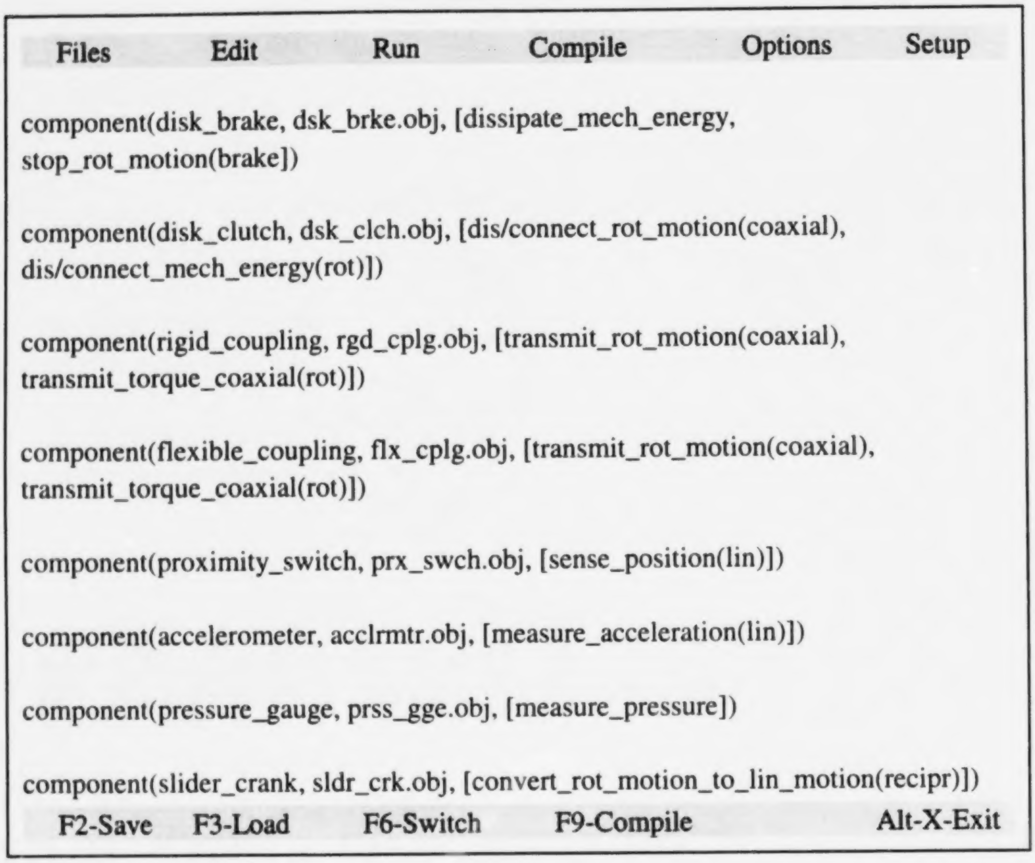
```
component(hydraulic_drive_(gear), hyd_gear.obj, [convert_rot_motion_to_lin_motion,
convert_kin_energy_to_pot_energy, supply_force])

component(cable_hoist, cabl_hst.obj, [convert_rot_motion_to_lin_motion, pull,
transfer_force(steady)])
component(chain_hoist, chn_hst.obj, [convert_rot_motion_to_lin_motion,
convert_kin_energy_to_pot_energy, supply_force])

component(hydraulic_drive_(gear), hyd_gear.obj, [convert_rot_motion_to_lin_motion,
convert_kin_energy_to_pot_energy, supply_force])

component(hydraulic_drive_(vane), hyd_gear.obj, [convert_rot_motion_to_lin_motion,
convert_kin_energy_to_pot_energy, supply_force])

component(hydraulic_drive_(pstn), hyd_gear.obj, [convert_rot_motion_to_lin_motion,
convert_kin_energy_to_pot_energy, supply_force])

component(pneumatic_drive, pnum_drv.obj, [convert_rot_motion_to_lin_motion,
convert_kin_energy_to_pot_energy, supply_force])

component(spur_gear_drive, spr_gear.obj, [transmit_power(rot), transmit_torque(rot),
adjust_rot_speed])
```

Figure D-1 (Continued)



```
component(bevel_gear_drive, bvl_gear.obj, [transmit_power(rot), transmit_torque(rot),
adjust_rot_speed])

component(helical_gear_drive, hel_gear.obj, [transmit_power(rot), transmit_torque(rot),
adjust_rot_speed])

component(worm_gear_drive, wrm_gear.obj, [transmit_power(rot), transmit_torque(rot),
adjust_rot_speed, make_ang_offset, stop_motion])

component(rack_and_pinion, rack_pin.obj, [convert_rot_motion_to_lin_motion,
transmit_power(rot)])

component(power_screw, pwr_scrw.obj, [convert_rot_motion_to_lin_motion,
transmit_power(rot), stop_motion(d)])

component(belt_drive, belt_drv.obj, [adjust_rotational_speed, transmit_power(rot)])

component(chain_drive, chn_drv.obj, [adjust_rotational_speed, transmit_power(rot)])

component(differential_pulley, dif-puly.obj, [redirect_lin_motion,
adjust_lin_displacement, adjust_lin_speed])
```

Figure D-1 (Continued)


Files
Edit
Run
Compile
Options
Setup
F2-Save
F3-Load
F6-Switch
F9-Compile
Alt-X-Exit

```
component(disk_brake, dsk_brke.obj, [dissipate_mech_energy,
stop_rot_motion(brake])

component(disk_clutch, dsk_clch.obj, [dis/connect_rot_motion(coaxial),
dis/connect_mech_energy(rot)])

component(rigid_coupling, rgd_cplg.obj, [transmit_rot_motion(coaxial),
transmit_torque_coaxial(rot)])

component(flexible_coupling, flx_cplg.obj, [transmit_rot_motion(coaxial),
transmit_torque_coaxial(rot)])

component(proximity_switch, prx_swch.obj, [sense_position(lin)])

component(accelerometer, acclrmtr.obj, [measure_acceleration(lin)])

component(pressure_gauge, prss_gge.obj, [measure_pressure])

component(slider_crank, sldr_crk.obj, [convert_rot_motion_to_lin_motion(recipr)])
```

Figure D-1 (Concluded)

APPENDIX E

TEST PROBLEMS TO DETERMINE OPTIMUM VALUES OF THE GENETIC PARAMETERS

For each problem the following are presented: a nomenclature of the variables of the problem, the equality and inequality constraint sets of the problem, and the penalty function to be minimized (zeroed) by the Genetic Algorithm.

TEST PROBLEM 1: *DESIGN OF A SPUR GEAR SET* (from Shigley 1986)

Nomenclature:

SR:	Speed Ratio (Input to Output)
IS:	Input Speed(rpm)
OS:	Output Speed(rpm)
TP:	Transmitted Power(w)
NPT:	Number of Pinion Teeth
NGT:	Number of Gear Teeth
ML	Module(m)
FW:	Face Width(m)
CD:	Centre Distance(m)
AT:	Applied Torque(N-m)
TL:	Transmitted Load(N)
MBS:	Maximum Bending Stress(Pa)
AlS:	Allowable Stress(Pa)
K_v:	Lewis Form Factor
Y:	Velocity Factor

Equality Constraints:

$$AT = \frac{TP}{IS}$$

$$CD = ML \,.\, NPT \,.\, \frac{1+SR}{2}$$

$$TL = \frac{TP \,.\, 60}{ML \,.\, NPT \,.\, IS \,.\, \pi}$$

$$OS = IS \,/\, SR$$

$$SR = \frac{NGT}{NPT}$$

$$MBS = \frac{TL}{k_v \,.\, FW \,.\, ML \,.\, Y}$$

$$k_v = \frac{360}{ML \,.\, NPT \,.\, IS \,.\, \pi + 360}$$

$$Y = -\,1.8 \times 10^{-8} \,.\, NPT^4 + 4.6 \times 10^{-6} \,.\, NPT^3 - 4.5 \times 10^{-4} \,.\, NPT^2 + 0.02 \,.\, NPT + 0.05$$

Inequality Constraints:

$CD - 0.4 \leq 0$

$0.2 - CD \leq 0$

$SR - 8.0 \leq 0 \quad (SR > 0)$

$FW - 16\ ML \leq 0$

$9\ ML - FW \leq 0$

$MBS - AIS \leq 0$

Penalty Function:

$$P.F. = [\max\,(0, \frac{CD-0.4}{0.3})]^2 + [\max\,(0, \frac{0.2-CD}{0.3})]^2 + [\max\,(0, \frac{SR-8}{4})]^2 + [\max\,(0, \frac{FW-16ML}{12.5ML})]^2 + [\max\,(0, \frac{9ML-FW}{12.5ML})]^2 + [\max\,(0, \frac{MBS-AIS}{MBS})]^2$$

TEST PROBLEM 2: *DESIGN OF A HELICAL COMPRESSION SPRING* (From Juvinall and Marshek 1991; Oberg and Jones 1971)

Nomenclature:

SAL:	Spring Axial Load (lb)
SWD:	Spring Wire Diameter (in)
COD:	Coil Outside Diameter (in)
STD:	Spring Total Deflection (in)
NAC:	Number of Active Coils
SSS:	Spring Shear Stress (psi)
MAS:	Maximum Allowable Stress (psi)
SFL:	Spring Free Length (in)
HSI:	Helical Spring Index
SWF:	Spring Wahl Factor
UTS:	Ultimate Tensile Stress (psi)

Equality Constraints:

$$STD = \frac{8 \times NAC \times SOD^{3} \times SAL}{TME \times SWD^{4}}$$

$$HSI = \frac{SOD}{SWD}$$

$$SWF = \frac{4 \times HSI - 1}{4 \times HSI - 4} + \frac{0.615}{HSI}$$

$$SSS = \frac{8 \times SWF \times SOD \times SAL}{\pi \times SWD^{3}}$$

Inequality Constraints:

$SSS \leq MAS$

$SFL / SOD \leq 4.0$

$SSS \leq 0.32 \times UTS$ [Based on a fatigue life of 10^6 fluctuations]

Penalty Function:

$$P.F. = [Max\ (0,\ \frac{SSS\ -\ MAS}{SSS})]^{2} + [Max\ (0,\ \frac{SFL\ -\ 40 \times SOD}{SFL})]^{2}$$

$$+ [Max\ (0,\ \frac{SSS\ -0.32 \times UTS}{SSS}]^{2}$$

TEST PROBLEM 3: *DESIGN OF A ROTATING SHAFT UNDER COMBINED LOAD*

(From Oberg and Jones 1971)

Nomenclature:

SOD:	Shaft Outside Diameter (mm)
SID:	Shaft Inside Diameter (mm)
HSF:	Hollow Shaft Factor
MSS:	Maximum Shearing Stress (N/mm^2)
MAS:	Maximum Allowable Shearing Stress (N/mm^2)
BMF:	Bending Moment Factor (Combined Shock and Fatigue)
TMF:	Torsional Moment Factor (Combined Shock and Fatigue)
MBM:	Maximum Bending Moment (N-mm)
MTP:	Maximum Transmitted Power (milliwatts)
SRS:	Shaft Rotational Speed (rpm)
TSD:	Torsional Shaft Deflection (degrees)
LSD:	Linear Shaft Deflection (mm)
LOS:	Length of Shaft (mm)
SME:	Shaft's Module of Elasticity (N/mm2)
TME:	Torsional Module of Elasticity (N/mm2)

Equality Constraints:

$$MSS = \frac{5.1 \times HSF^{3}}{SOD^{3}} \sqrt{(BMF \times MBM)^{2} + (TMF \times MTM)^{2}}$$

$$MTM = \frac{9.55 \times MTP}{SRS}$$

$$HSF = \sqrt[3]{1 - (\frac{SID}{SOD})^{4}}$$

$$LSD = \frac{32 \times MBM \times LOS^{2}}{\pi \times SME \times (SOD^{4} - SID^{4})}$$

$$TSD = \frac{32 \times MTM \times LOS}{\pi \times TME \times (SOD^{4} - SID^{4})}$$

Inequality Constraints:

MSS ≤ MAS	
ASD ≤ LOS / (20 × SOD)	[Based on maximum allowable deflection of one degree per 20 diameters of the shaft]
LSD ≤ LOS / 1200	[Based on maximum allowable deflection of one mm per 1200 mm length of the shaft]

Penalty Function:

$$P.F. = [Max\ (0, \frac{MSS - MAS}{MSS})]^{2} + [Max\ (0, \frac{20 \times SOD \times ASD - LOS}{ASD})]^{2}$$

$$+ [Max\ (0, \frac{1200 \times LSD - LOS}{LSD})]^{2}$$